Institut für Visualisierung und Interaktive Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Untersuchung der Anforderungen und Workflows für den Austausch von Assets zwischen Content-Creation und Echtzeitrendering

Charlene Masri

Studiengang:	B. Sc. Medieninformatik
Prüfer/in:	Prof. Dr. Thomas Ertl
Betreuer/in:	Dr. Guido Reina, Michael Becher, M.Sc.
Beginn am:	6. September 2021
Beendet am:	6. März 2022

Kurzfassung

An der Produktion von Echtzeitrenderingprojekten sind meist mehrere Entwickler:innen und Artists beteiligt, die 3D-Inhalte kreieren. Diese Inhalte werden in verschiedenen DCC-Anwendungen erstellt und schließlich in das Echtzeitrenderingprogramm importiert. Der Austausch der 3D-Assets ist häufig ein verlustbehafteter Schritt in der Produktion, der oft wiederholt werden muss. Das erfordert einen definierten Umgang mit den Export- und Importschnittstellen der Programme.

In dieser Arbeit wird ein Kriterienkatalog für den Austauschprozess von Assets zwischen 3D-Anwendungen erarbeitet und angewendet. Der Kriterienkatalog basiert auf Interviews zu Nutzungsanforderungen und Workflows von 3D-Entwickler:innen aus verschiedenen Branchen und Teilen der Produktionspipeline. Zusätzlich wird die NVIDIA Omniverse Plattform zur Kollaboration an 3D-Projekten vorgestellt und ein Omniverse Connector für das Visualisierungsframework MegaMol entwickelt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AEC Architecture, Engineering and Construction. 29

BIM Building Information Modeling. 40

CAD Computer Aided Design. 7

CES Consumer Electronics Show. 13

CPU Central Processing Unit. 32

DCC Digital Content Creation. 3, 11

DOM Document Object Model. 27

FBX FilmBox Dateiformat. 5

FK Forward Kinematics. 22

GDC Game Developers Conference. 15

glTF GL Transmission Format. 5

GPU Graphics Processing Unit. 19, 32, 40

HDRP High-Definition Render Pipeline. 20

IFC Industry Foundation Classes Dateiformat. 41

IGDA International Game Developer Association. 11

IK Inverse Kinematics. 22

LOD Level of Detail. 25, 36

MDL NVIDIA Material Definition Language. 29

MTL Material Template Library. 25

NURBS Nonuniform Rational B-Splines. 17

PBR Physically Based Rendering. 19

SDK Software Development Kit. 25

STEP Standard for the Exchange of Product Model Data. 41, 42

TBB Thread Building Blocks. 32

TD Technical Director. 11

URP Universal Render Pipeline. 20

USD Universal Scene Description. 5

USDA Universal Scene Description Textformat. 32, 59

VFX Visual Effects. 15

VM Virtuelle Maschine. 32

VR Virtual Reality. 35

XML Extensible Markdown Language. 26

1 Einleitung

Eine Crunchtime meint den Zeitraum vor der Abgabe eines Projekts, der aufgrund von Verzug im Zeitplan eine außergewöhnlich hohe Beanspruchung der Projektteilnehmenden erfordert. Die letzten fehlenden Features müssen priorisiert und bestehende Fehler unbedingt vor der Abgabefrist behoben werden. Dafür arbeiten die Entwickler:innen mit Hochdruck auch außerhalb der üblichen Arbeitszeiten.

In der im Oktober 2021 erschienenen Developer Satisfaction Survey der International Game Developer Association (IGDA) gaben ein Drittel der Befragten an, ihr Job beinhalte „Crunchtime“. Weitere 22% behaupteten, ihre Arbeitsstelle verlange Überstunden. Das Problem, dass die Produktion mehr Zeit braucht, als eingeplant wurde, betrifft demnach circa die Hälfte aller Befragten [IGDA21]. Es lohnt sich also, einen Blick auf bestehende Workflows zu werfen und zu versuchen, existierende Engpässe zu erkennen und Lösungen zu finden.

In der Gameentwicklung und Entwicklung von 3D-Inhalten haben viele Faktoren Einfluss auf die Gestaltung des Workflows. Um die spezifischen Anforderungen von 3D-Projekten zu bedienen, sind meistens mehrere Anwendungen und Plugins involviert, deren Auswahl entscheidend für den Workflow ist.

1.1 Die Produktionspipeline

Innerhalb einer Produktion durchlaufen 3D-Assets für gewöhnlich unterschiedliche Produktionsschritte, manche davon in verschiedenen 3D-Anwendungen. Unter einem Asset versteht man ein Objekt, das in einer 3D-Szene eingesetzt werden kann. Es kann außerhalb oder innerhalb einer Echtzeitengine erstellt werden. In Unity3d kann das z. B. ein 3D-Modell, Audio-Dateien, ein Bild, eine Render-Textur etc. sein [Uni22d].

Diesen Datenfluss nennt man Produktionspipeline bzw. Assetpipeline, Game-Assetpipeline oder Contentpipeline. Das Aufkommen von verschiedenen Aufgaben, wie beispielsweise Modellieren, Animieren oder Texturieren, verlangt verschiedene Expert:innen, die unterschiedliche Programme zum Erstellen von 3D-Assets verwenden, sogenannte Digital Content Creation (DCC)-Tools. Viele davon sind nicht ausschließlich auf das Erzeugen von Echtzeitrenderinginhalten beschränkt, sondern finden vor allem im Erstellen von Offline-Renderings Gebrauch.

Verschiedene Schritte innerhalb einer Produktionspipeline sind voneinander abhängig und Assets werden untereinander und zwischen den Programmen oft ausgetauscht. Um den Datenaustausch zu regeln, sind manchmal tiefgreifende Kenntnisse gefragt. Manche Projekte erfordern sogar Pipeline Technical Directors (TDs), die für den reibungslosen Ablauf der Assetpipeline zuständig sind und Lösungen für technische Probleme finden, sobald sie aufkommen [ftr21].

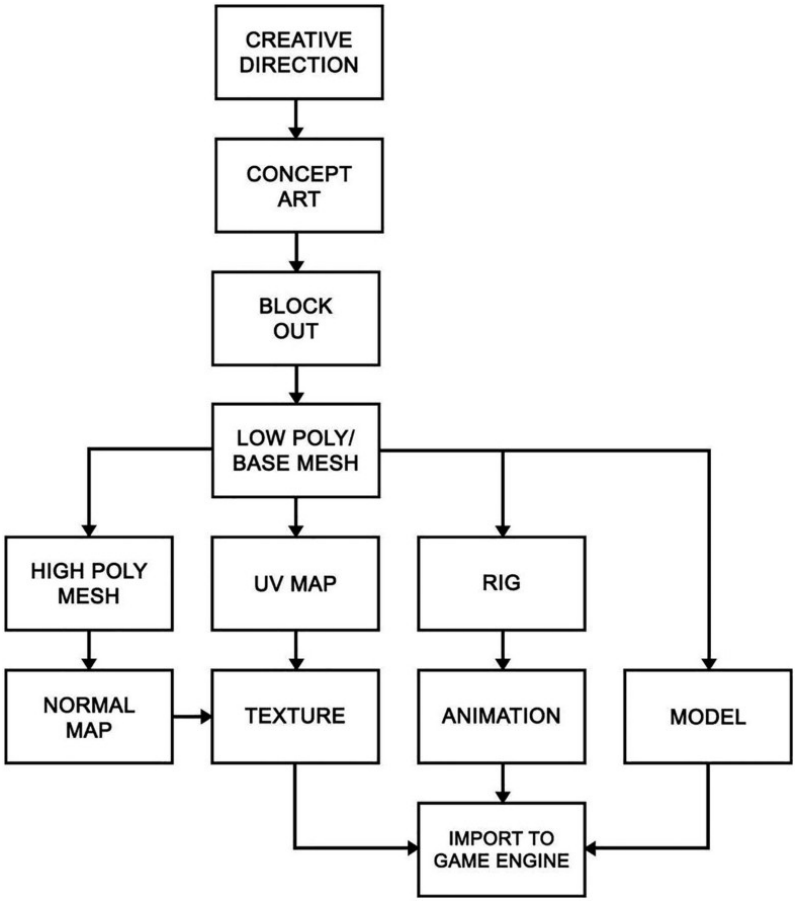

Abbildung 1.1: Allgemeine Game Development Pipeline nach Villanueva [Vil22]

Abbildung 1.1 zeigt ein Beispiel einer Produktionspipeline für die Spielentwicklung. Zu sehen ist ein Flowchart, das die Abhängigkeiten der Erstellung verschiedener 3D-Assets von Konzept bis zum Import in die Echtzeitengine darstellt.

1.2 Motivation

Echtzeitrendering bringt abseits der Pipelineengpässe auch noch eine andere Form von Engpässen mit sich: die Hardwaregrenzen. Diese Form von Rendering erfordert optimierte Algorithmen und Datenstrukturen für die zu visualisierenden Daten. 3D-Objekte, Shader, Animationen und weitere Assets sind in Game Engines deshalb auf gewisse Standards beschränkt, die für das Offline-Rendern nicht gelten.

Gleichzeitig werden die Assets in nicht für Echtzeitrendering optimierten Programmen mit umfangreicheren Möglichkeiten erstellt. Wird es also den Nutzer:innen überlassen, nur den Content in den DCC-Anwendungen zu erstellen, der von Game Engines visualisierbar ist? Es gibt viele Austauschlösungen zwischen 3D-Applikationen, die versuchen, die Assets möglichst akkurat zu übersetzen. Dieser Prozess ist häufig fehleranfällig und benötigt Pflege und Zeit, um sicherzustellen, dass die Assets zufriedenstellend und korrekt übertragen werden.

Im dieser Arbeit wird der Austausch von 3D-Assets behandelt. Das Ziel ist es einen Kriterienkatalog zu erstellen, der zum Kategorisieren und Bewerten von Datenaustauschszenarien zwischen 3D-Programmen dient. Außerdem wird die NVIDIA Omniverse-Plattform für die Kollaboration an 3D-Projekten untersucht. Im Januar 2022 hatte sie auf der CES ihren offiziellen Release [NVI22a], zuvor war eine Beta-Version verfügbar.

1.3 Aufbau der Arbeit

Zunächst werden verwandte Arbeiten vorgestellt, die in Verbindung mit der Literaturrecherche für diese Arbeit gefunden wurden und diese Arbeit beeinflusst haben. Danach werden einige Bestandteile von 3D-Szenen vorgestellt, um grundlegende Begriffe der auszutauschenden Assets zu klären. In Kapitel 4 werden verschiedene Dateiformate, die zum Austausch von 3D-Assets verwendet werden, vorgestellt. Diese beiden Kapitel dienen dazu, die Grundlagen für die Untersuchung von 3D-Austauschprozessen zu schaffen.

Weiterführend wird ein neuer Ansatz zur Kollaboration an 3D-Projekten, die NVIDIA Omniverse-Plattform, vorgestellt und die Entwicklung einer Importschnittstelle zu NVIDIA Omniverse präsentiert.

Im Zuge dieser Arbeit wurden Interviews mit verschiedenen 3D-Artists und 3D-Entwickler:innen durchgeführt, um zu erfahren, welche Probleme und Besonderheiten bei dem Austausch von 3D-Assets zwischen Programmen anfallen. Diese Interviews helfen dabei, Informationen über verschiedene 3D-Content-Pipelines und Anforderungen an die 3D-Anwendungen von Nutzer:innen zu sammeln. Die Umsetzung und Ergebnisse der Interviews werden in Kapitel 6 aufgeführt.

Auf Grundlage der Interviewergebnisse wurde ein Kriterienkatalog für Austauschprozesse entwickelt, der in Kapitel 7 präsentiert wird. Schließlich wird der Kritetrienkatalog auf zwei exemplarische Austauschszenarien im folgenden Kapitel angewendet und die Szenarien werden evaluiert. Die Arbeit endet mit einem Fazit aus der Anwendung des Kriterienkatalog und einem Kommentar zu NVIDIA Omniverse.

2 Verwandte Arbeiten

Die Begriffe Game-Development-Pipeline oder 3D-Asset-Pipeline kommen vor allem in Büchern und Vorträgen über Spielentwicklung und 3D-Animationen der letzten 20 Jahre vor. Viele davon sind einem bestimmten Anwendungsfall gewidmet. So geht es zum Beispiel in Villanuevas „Beginning 3D Game Assets Development Pipeline“ um den Export von Assets von Maya zu Unity [Vil22]. Das Buch erläutert die Erstellung und den Austausch von Assets der beiden spezifischen Programme und erklärt Schritte der Game-Asset-Pipeline. Von manchen Unternehmen gibt es Vorträge und Artikel über die Produktionspipelines eigener Projekte. Beispielsweise stellte Guerilla Games in ihrem Vortrag auf der Game Developers Conference (GDC) „Creating a Tools Pipeline for Horizon: Zero Dawn“ eigene kreierte Tools für die Entwicklung ihrer Spiele vor [Dan17].

Aber nicht nur aus der Game-Branche gibt es Literatur, die die 3D-Asset-Produktion behandeln, sondern auch in der 3D-Animationsbranche präsentieren verschiedene Unternehmen Pipelines und Standards ihrer Workflows. Sie benutzen die gleichen DCC-Anwendungen oder zumindest Anwendungskonzepte. Der Austausch von 3D-Assets prägt auch die Branche und eine performante Echtzeitvorschau der 3D-Szenen ist in ihrem Designprozess ebenso von Relevanz [GJJ16]. Pixar Animation Studios und andere Mitwirkende der Academy Software Foundation tragen beispielsweise dazu bei, 3D-Standards der Visual Effects (VFX)-Branche zu entwickeln und OpenSource-Projekte, die auch die Produktionspipeline von Echtzeitrenderings beeinflussen, offenzulegen [Pix21a; The18].

Die Dokumentationen und Manuals von unterschiedlichen 3D-Anwendungen bieten oftmals Informationen zu ihren Import- oder Exportlösungen. Vor allem kostenlose Anwendungen haben große detaillierte Dokumentationen, z.B. Unity, Unreal Engine und Blender. Sie schildern Asset Workflows und Schnittstellen zu bestimmten Austausch-Dateiformaten [Blea; Epi22b; Uni22a]

In James Lears „The Video Game Asset Pipeline - A pattern Approach“ wird versucht, Game Asset Pipelines mit einer Mustersprache zu formalisieren [Lea20]. Seine Arbeit hat vor allem konzeptuell dazu beigetragen, den Kriterienkatalog dieser Arbeit in Form von Problemfragen zu entwerfen. Während der Literaturrecherche für diese Arbeit konnte keine Literatur zu Kriterien von 3D-Asset-Pipelines von Echtzeitrenderingprojekten gefunden werden, weshalb Interviews mit zwölf Nutzer:innen von 3D-Anwendungen aus verschiedenen Branchen durchgeführt wurden.

3 Bestandteile von 3D-Szenen und ihre Repräsentationen

3D-Projekte können aus den unterschiedlichsten Assets bestehen. In Videospielen existieren zahlreiche Assets, die abstrakter sind als ein 3D-Objekt oder eine Lichtquelle. Ein Besipiel dafür ist eine Player Start Location. Solche Bestandteile sind wichtig für die Spielführung und Interaktion, werden aber erst in der Game Engine kreiert. Dieses Kapitel beschränkt sich nur auf übliche Elemente, die in DCC-Anwendungen erstellt werden, um in Echtzeitrenderern dargestellt zu werden. Diese Aufzählung ist vor allem ein Versuch, Begrifflichkeiten von 3D-Anwendungen und 3D-Assets zusammenzufassen und für die weiteren Kapitel aufzuschlüsseln.

3.1 Geometrien

Für die Oberflächen von 3D-Geometrien gibt es zwei übliche Repräsentationsansätze: parametrische und polygonale Oberflächen. Parametrische Oberflächen werden mithilfe parametrischer Kurven erzeugt [FDFH96, S. 471]. Polygonale Oberflächen bestehen aus flachen Polygonen, die, um organische Oberflächen darzustellen, kurvige Flächen lediglich annähern. Man unterscheidet zwischen CAD-Modellen, die aus parametrischen Oberflächen , z.B. aus Nonuniform Rational B-Splines (NURBS), bestehen und Meshes, die aus Polygonen bestehen. In Abbildung 3.1 sieht man ein CAD-Modell und das entsprechende in Dreiecke unterteilte Mesh.

3.1.1 CAD-Modelle

Diese Modelle kommen aus CAD-Anwendungen, die in Branchen genutzt werden, in denen eine exakte Beschreibung der Geometrie qualitativ notwendig ist, zum Beispiel Automobilbranche, Produktdesign, Architektur, Maschinenbau etc.

Häufig beinhalten sie Metadaten wie Herstellungsinformationen [Pix21c]. Sie benötigen wegen ihrer kompakten Beschreibung im Vergleich zu Meshes nicht so viel Speicherplatz und haben den Vorteil, dass sie in jeder Skalierung hochaufgelöst sind. Sie eignen sich allerdings nicht für Echtzeitrenderer, da die Berechnungen für die Auswertung der Oberflächen sehr aufwändig sind. Daher konvertiert man die Modelle zu Meshes, wenn man sie in Echtzeitanwendungen darstellen möchte.

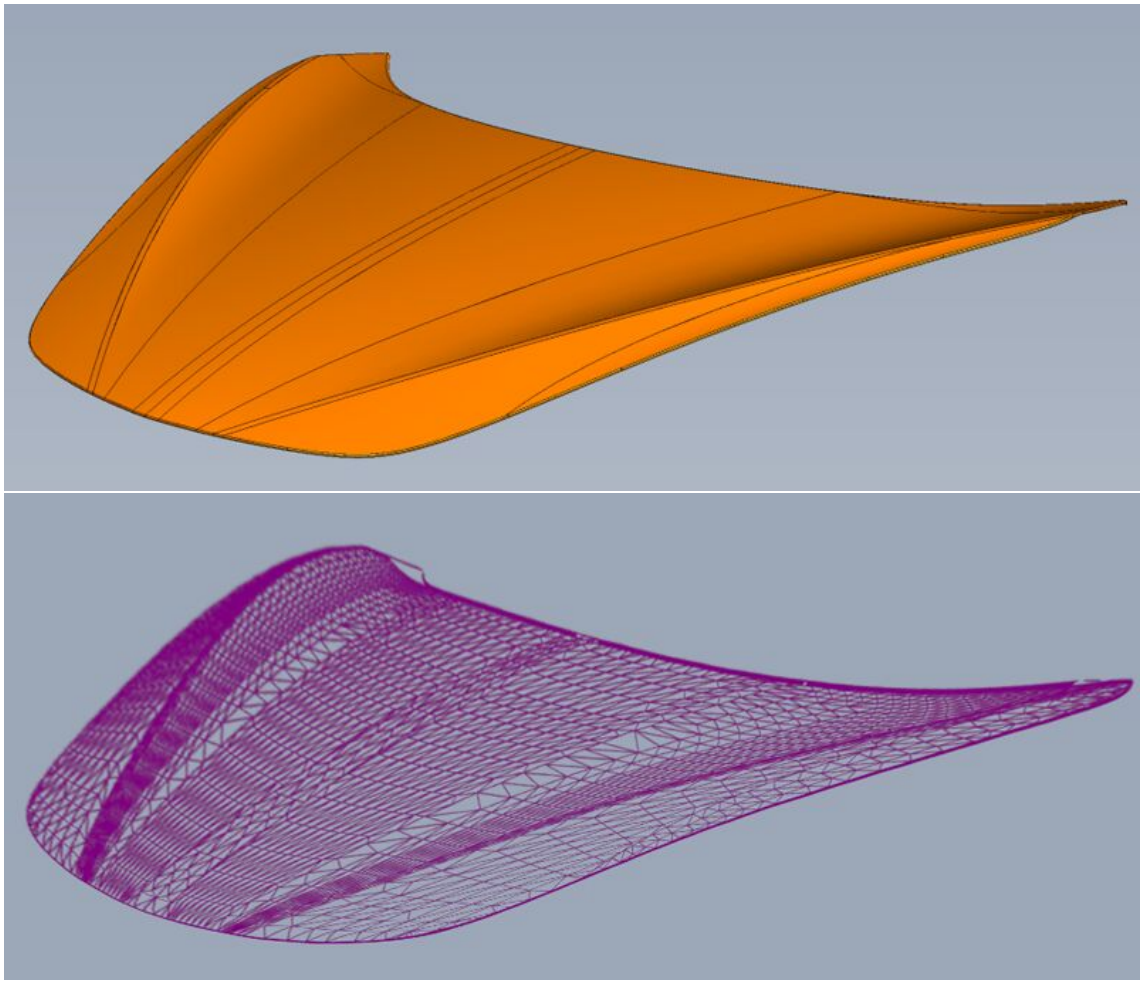

Abbildung 3.1: Gegenüberstellung von CAD-Modell (oben) und trianguliertem Mesh (unten) [Epi22b]

3.1.2 Meshes

Meshes bestehen aus Polygonen, deren Eckpunkte Vertices heißen. In vielen DCC-Tools, z.B. in Blender, können Meshes aus verschiedenen Primitiven zusammengesetzt sein, gängig sind jedoch Quads (Vierecke) [Ble22a]. Das ist in Abbildung 3.2 links zu sehen. In Echtzeitengines sind üblicherweise Tris (Dreiecke) die zugrundeliegenden Polygone.

Vertices können über verschiedene Attribute verfügen: zum Einen hat ein Vertex eine Position in einem 3D-Koordinatensystem, zusätzlich kann er aber noch Attribute wie einen Normalenvektor, Texturkoordinaten, eine Farbe etc. besitzen. Diese Attribute werden unter Anderem vom Shader verwendet, um Lichtberechnungen durchzuführen und Farben zuzuweisen.

Möchte man auf die Oberfläche einer Geometrie Texturen oder andere Materialinformationen mappen, die aus 2D-Formaten ausgelesen werden, benötigt sie Texturkoordinaten (auch UV-Koordinaten genannt). Dafür wird jedem Vertex eine 2D-Texturkoordinate zugewiesen, aus denen

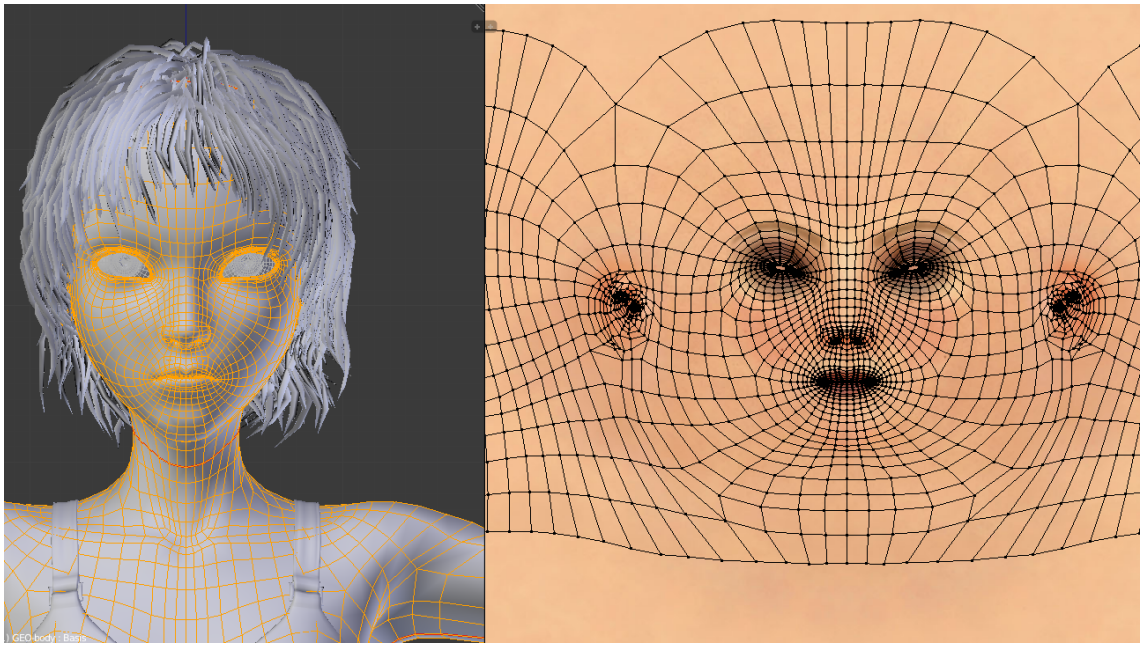

Abbildung 3.2: Mesh aus Blender (links) mit UV-Map (rechts) [Blec]

in der Menge eine UV-Map entsteht. Auf dieser Grundlage können die 2D-Shadinginformationen auf die Oberflächen gelegt werden. In Abbildung 3.2 sieht man rechts eine entsprechende UV-Map mit eingezeichneten Edges.

3.2 Shading

Damit ein Renderer weiß, wie die Objekte darzustellen sind, brauchen die Geometrien eine Oberflächenbeschreibung (Materialien) und Lichter, von denen sie beleuchtet werden. Die Berechnungen dafür führen Shader auf der GPU durch. Ein Shader ist ein Programm, das Per-Vertexoperationen durchführt und die Farbigkeiten der Pixel ermittelt [OCo17].

3D-Renderer können mit verschiedenen bereits definierten Shading-Modellen ausgestattet sein. Neben den klassischen Shading-Modellen wie Lambert oder Phong gibt es auch moderneres Physically Based Rendering (PBR). PBR-Techniken versuchen reale Illumination für das Erstellen fotorealistischer Umgebungen zu imitieren oder anzunähern [Bab]. Es hängt von dem Renderer, den Shading-Modellen und Illuminationsmodellen ab, welche Lichtquellen und Materialien erstellbar sind und wie sie interpretiert werden.

3.2.1 Lighting

Licht kann sehr wichtig für den Look von einer Szene sein und verschiedene Techniken prägen die finale Lichtabmischung in den einzelnen Renderern. Das ganze Spektrum von Lighting abzudecken, würde den Rahmen dieser Arbeit sprengen. Selbst ein 3D-Programm kann unterschiedliche Render Engines haben. Es gibt also eine Menge unterschiedlicher Lichtkonfigurationen.

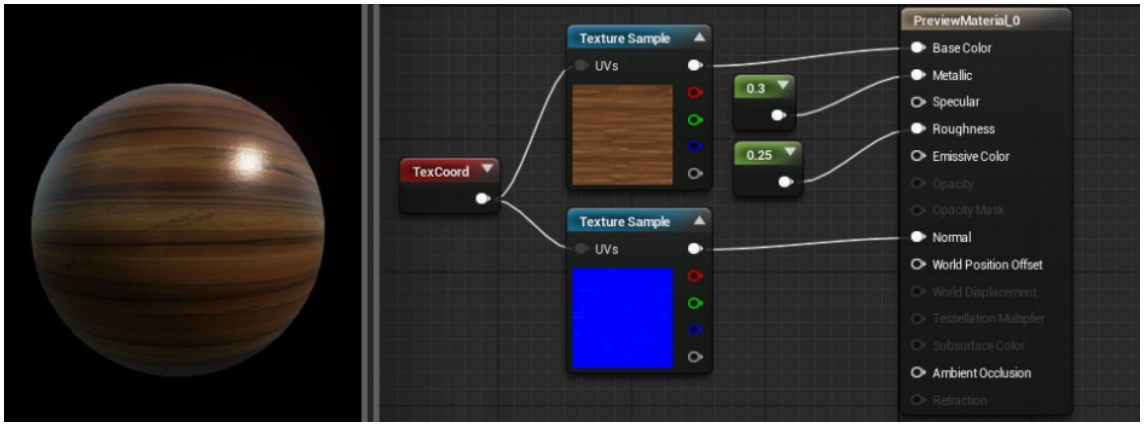

Abbildung 3.3: Node-Graph eines Materials in Unreal Engine [Epi22c]

Generell könnte jede geometrische Form eine Lichtquelle sein, die in einer Umgebung platziert ist und deren Material Licht emittiert. Es ist vom Renderer abhängig, welche Lichtquellen und Lichtberechnungen möglich sind. Es gibt allerdings etablierte Lichtquellen, die man in einigen 3D-Anwendungen findet: Point-Lights, Spot-Lights und gerichtete (directional) Lichtstrahlen. Point-Lights strahlen in alle Richtungen von einem Punkt aus, Spot-Lights strahlen kegelförmig aus, gerichtete Lichtstrahlen imitieren Lichtquellen, die weit weg sind und somit nahezu parallele Strahlen haben z.B. die Sonne [Uni22b]. Es gibt aber auch andere Lichtquellen, z.B. ambientes Licht.

Position oder Ausrichtung, Intensität, Temperatur und Farbe sind übliche Attribute. Außerdem können Lichtquellen einen Radius oder eine Länge haben. Für manche Lichtquellen kann man definieren, wie das Licht abfällt, beispielsweise für ein Spotlight, um weichere Beleuchtungskanten darzustellen. Die Attribute sind abhängig von Art und Funktionsweise der Lichtquelle.

Des Weiteren haben Lichter meist rendererspezifische Attribute, beispielsweise die maximale Anzahl an „Bounces“ für indirekte Beleuchtung im Arnold-Renderer in Maya [Aut21] oder die Mobilität in Unreal Engine [Epi22a]. Die Mobilität kann statisch, stationär oder dynamisch sein, denn in Real-Time-Engines kann die Lichtinformation von statischen bzw. stationären Lichtern auf Light Maps gebacken werden, um die Anwendung performanter zu machen. Manche Lichteinstellungen werden global für die Szene oder das ganze Projekt konfiguriert.

3.2.2 Materials

Mit Materialien legt man fest, wie die Oberfläche eines Objektes erscheint und auf welche Art das Licht gebrochen wird, das auf die Oberfläche trifft. Häufig gibt es Node-basierte Material-Editoren in den Programmen, in denen man Shaderoperationen hinzufügen kann und Materialparameter befüllt, so auch in Unreal Engine, Unity3d (Universal Render Pipeline (URP) und High-Definition Render Pipeline (HDRP)), Maya und Blender [Aut22; Ble22b; Epi22c; Uni22c]. Welche Materialparameter oder Nodes existieren, variiert von Anwendung zu Anwendung. In Abbildung 3.3 ist ein Beispiel eines Material-Editors in Unreal Engine. Der Editor stellt in dem Beispiel simple PBR-Parameter, z.B. Metallic und Roughness, bereit (rechter Node) [Kar13].

Um ein Material zu editieren, weist man also den verfügbaren Parametern, wenn sie für das ganze Objekt gleichermaßen gelten, konstante Werte zu, oder arbeitet mit 2D-Maps, die auf die UVs einzelner Modelle abgebildet werden (siehe Abbildung 3.3). Das sind beides Ansätze, bei denen die Parameter lediglich aus dem Speicher ausgelesen werden. Man kann auch dynamischere Materialien erstellen, indem man verschiedene Werte berechnen lässt. Es gibt z.B. prozedurale Materialien, die generierte Texturen verwenden [Uni17]. Diese Materialien werden häufig auf mehrere Objekte angewendet, ohne dass sie immer gleich aussehen, denn die Texturparameter können pro Objekt variiert werden. Die Erstellung von prozeduralen Materialien kann zeitaufwändig sein und, wenn während Laufzeit berechnet, können sie die Shaderkomplexität erhöhen.

Materialien können sehr vielseitig sein. Sie können sogar die tatsächliche Form und Position eines 3D-Objektes beeinflussen. Insgesamt deckt dieser Abschnitt nur einen Bruchteil des Themas ab. Die Vielzahl an 3D-Renderern und unterschiedlichen Techniken macht es schwierig eine einheitliche Übersicht abzuleiten, denn die Material- bzw. Shader-Editoren in DCC-Programmen können komplexere Materialausdrücke und Nodes beinhalten.

3.2.3 Texturen

Wie in oberen Abschnitten bereits erwähnt, sind Texturen 2D-Bilder, die über eine Geometrie gelegt werden. Anhand von UV-Koordinaten eines Objektes werden bestimmte Bildbereiche auf bestimmte Oberflächenbereiche übertragen. Unterschiedliche Materialparameter können durch Texturen ausgedrückt werden. Manche Parameter benötigen nur einen Farbkanal, z.B. Metallness, andere benötigen sogar nur ein binäres Bild, z.B. eine Opacity Mask.

Für die Optimierung der Assets werden manche Texturen "gebacken". Dabei werden Informationen eines High-Poly Meshes auf eine Low-Poly Version übertragen [Ado21]. Diesen Workflow sieht man auch in Villanuevas Game Development Pipeline in Abbildung 1.1 für das Erstellen einer Normal-Map. Für manche Bakingprozesse, z.B. Ambient Occlusion, braucht man nicht notwendigerweise zwei Repräsentationen des Objektes [Ado19]. Das Resultat einer angewandten Normal-Map wird in Abbildung 3.4 dargestellt.

3.3 Animations

Im Gegensatz zu statischen Objekten verändern sich animierte 3D-Objekte in ihrer Position oder ihrer Form über eine bestimmte Zeit. Auch andere 3D-Assets können animiert werden, z.B. Kameras, Materialien oder Lichter. Diese Arbeit beschränkt sich jedoch auf die Animation von Geometrien. Blender fasst den Begriff Animation zusammen in Bewegung eines ganzen Objektes, Deformation von Objekten und Vererbung von Animation von anderen Objekten [Ble22c].

Eine Animation wird üblicherweise anhand von Keyframes definiert, die an „wichtigen“ Zeitpunkten gesetzt werden und einen bestimmten Wert für einen Parameter eines Objektes beinhalten [PAM10, S. 418]. Zwischen zwei Keyframes werden die Werte mit einer ausgewählten Methode interpoliert. Das ergibt eine Animationskurve. Um Animationen zu erstellen, gibt es verschiedene Konventionen und Strukturen, die das Objekt kontrollieren können.

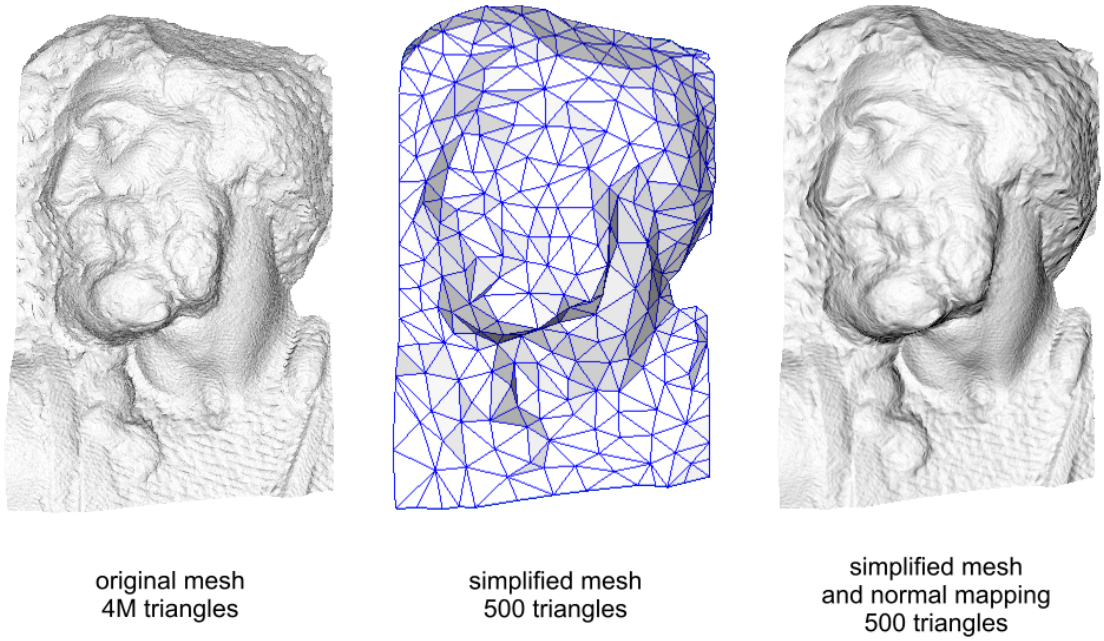

Abbildung 3.4: Baking von Normalen eines High-Poly Meshes auf ein Low-Poly mit Resultat (rechts) (Bild ursprünglich von Paolo Cignoni) [Ado21]

3.3.1 Rigging

Für ein 3D-Objekt kann ein unterliegendes Skelett modelliert werden, anhand dessen Animationen kreiert werden können, die auf das Objekt Einfluss nehmen. Skelette bestehen aus Joints, die hierarchisch strukturiert sind [PAM10, S. 427]. Die Transformation eines Joints hat unmittelbaren Einfluss auf andere Joints in der Hierarchie. Dabei gibt es zwei Techniken, die häufig in Kombination verwendet werden: Forward Kinematics (FK) ist die Bezeichnung des Einflusses, den die Transformation von Elternknoten in der Joint-Hierarchie auf Kindknoten hat. Umgekehrt beschreibt Inverse Kinematics (IK), wie die Transformation eines Kind-Joints, sich auf die Eltern-Joints auswirkt [Plu14; Rea]. Für die weitere Animation eines Skeletts müssen üblicherweise bestimmte Einschränkungen (Limit Constraints) definiert werden, die den Freiheitsgrad der Joints bzw. Bones bestimmen [PAM10, S. 430; Ble22d].

Joints können verschiedene Ausführungen und Eigenschaften in Animationsanwendungen haben. Blender benutzt beispielsweise den Begriff Bones, also eine Kombination von zwei Joints, um ein Skelett aufzubauen.

Die Zuweisung der Vertices auf die Joints, die Einfluss auf sie nehmen, nennt man Skinning. Dabei kann ein Vertex von einem oder mehreren Joints gemäß einer bestimmten Gewichtungen kontrolliert werden [PAM10, S. 430].

3.3.2 Constraints

Constraints können bestimmte Limitierungen für die Transformation eines Objektes festlegen, z.B. IK-Constraints in Blender, die die Bones auf eine bestimmte Bewegungsfreiheit limitieren. Man kann aber auch Objekte oder Bones durch Constraints miteinander verknüpfen und somit die Transformation eines Objektes an die Bewegung eines anderen Objektes binden. So vererbt ein Objekt die Transformation eines anderen auf eine festgelegte Weise [Ble22e].

3.3.3 Blend Shapes

Blend Shapes werden häufig auch Morph Targets oder Shape Keys genannt. Sie definieren ein Set an verschiedenen Verformungen eines Objektes. Das kann hilfreich für Animationen sein, bei denen die Erstellung eines komplexen Rigs nicht sinnvoll wäre, z.B. für Gesichtsausdrücke, da sie aus ganz vielen feinen Deformationen bestehen [Ble22f]. Die Deformationen können mit allen Modellierungstools erstellt werden, die das Programm zu Verfügung stellt, z.B. die Verdrehung oder Verbiegung eines ganzen Objektes oder die Verformung bestimmter Bereiche.

Man kopiert dafür das Basismesh, deformiert es und deklariert die deformierte Kopie als Zielform einer Blend Shape eines Objektes. Während der Animation wird zwischen den Basis- und Zielpositionen der Vertices interpoliert. Die Anzahl der Vertices sollte sich dabei nicht verändern [Ble22f].

4 Dateiformate für den Austausch von 3D-Assets

Es gibt viele Formate, die für den Austausch von 3D-Assets entwickelt wurden und auf unterschiedlichen Konzepten basieren. Im Folgenden werden einige populäre Austauschformate vorgestellt, die in der Games und DCC-Branche häufig Anwendung finden.

4.1 Autodesk FBX

FBX ist ein proprietäres Dateiformat für den 3D-Assetaustausch. Es wird besonders beim Austausch von Animationsdaten verwendet. FBX gehört zu Autodesk und wird daher vor allem von Autodesk-Anwendungen wie Maya, 3ds Max oder Motion Builder unterstützt. Aber auch Blender, Unreal Engine, Unity3d, Cinema4d und viele weitere prominente 3D-Programme können das FBX-Format einlesen oder ausspeichern. Um FBX zu integrieren, benötigt man das entsprechende geschlossene SDK [Hou19c] von Autodesk. Es gibt einen öffentlichen Programmer's Guide für das SDK und der Strukturen, die FBX zugrunde liegen [Aut]. Das Dateiformat kann sowohl eine ASCII- als auch eine binäre Version sein [Ble10].

Die Assets, die FBX enthalten kann, sind sehr vielfältig. In dem Scene Graph dieses Formates können Meshes, LODs, Materials, NURBS, Textures, Kameras, Lights, Constraints, Vertex-Cache-Animationen, Szeneneinstellungen, Transformationen, Marker für Motion Capturing, Skelette, Animationskurven und Posen repräsentiert werden [Aut]. Welche Attribute und Datenstrukturen umsetzbar sind, geht anhand von Beispielen und Beschreibungen aus dem oben erwähnten Programmer's Guide hervor.

FbxSurfaceMaterials basieren auf Lambert- und Phong-Obeflächenattributen, Texturen werden über den Dateinamen verlinkt und Lichtquellen haben eine eher simple Beschreibung und umfassen nicht alle Beleuchtungsmodelle von modernen Renderern [Aut; Hou19c].

4.2 Wavefront OBJ

Im Gegensatz zu Autodesk FBX-Dateien ist das OBJ-Format um Einiges simpler. Es wird trotz seines Alters von vielen 3D-Programmen unterstützt, denn es lässt sich leicht lesen und integrieren. OBJ eignet sich nicht für die Übertragung von Szenenhierarchien, Lighting oder Animationen, sondern hauptsächlich für 3D-Modelldaten [Hou19b].

Eine OBJ-Datei ist eine Textdatei, die Meshes, LOD-Level, Gruppierungen, NURBS-Oberflächen und -Kurven, Szeneneinstellungen wie die Up-Axis, und Materialien speichern kann [Bleb; Red]. Materialien werden in einer seperaten MTL-Datei gespeichert. Vertices können Positionen, Texturkoordinaten und Normalen besitzen [Bleb].

4.3 glTF

Die Khronos Group entwickelte das 3D-Asset-Format glTF, um einen effizienten Ausleseprozess zur Laufzeit zu ermöglichen [Khr22]. Daher wird es auch häufig für Webanwendungen benutzt [Hou19a]. glTF-Dateien bestehen aus einer glTF-JSON-Datei, die die Szenenhierarchie beschreibt und auf komplexere binäre, extern gespeicherte Assets verweist. Es gibt auch Möglichkeiten die Daten gesammelt in einer JSON-Datei oder einer binären Datei zu speichern [Blea]. Die gängigen Game Engines und DCC-Anwendungen haben glTF-Importer bzw. Exporter oder es existieren Erweiterung dafür.

Das Format kann unterschiedliche Assets beinhalten, darunter Meshes, Transformationen, Materialien, Texturen, Kameras, Lichter und Animationen wie Morph Targets und Skinned Meshes mit Skeletten [Blea; Khr21]. Die glTF-Spezifikation beschreibt die wesentlichen Konzepte des Formates und gibt an, welche Einschränkungen auf verschiedenen Datenstrukturen liegen.

glTF hat einen PBR-basierten Metallic/Roughness Material-Standard und unterstützt verschiedene Alphawertumsetzungen, bestimmte Texture-Maps und die Zweiseitigkeit von Flächen [Lin17]. Das Format kann durch benutzerdefinierte Erweiterungen angepasst werden [Khr21].

4.4 Alembic

Alembic wurde von Lucasfilm und Sony Pictures Imageworks als ein Dateiformat für den Austausch von komplexen Animationen und VFX Szenen entwickelt [PR 11]. Das Besondere an Alembic ist, dass es nativ keine Animationskurven oder andere prozedurale Animationsbeschreibungen unterstützt, sondern ein „Geometry Caching Format“ [Son16] ist. Das bedeutet, dass das Format die Transformation der gesamten Szene zeitweise speichert bzw. die gebackenen Animationsergebnisse cached.

Alembic kann verschiedene geometrische Repräsentationen speichern, darunter Meshes, Subdivision Surfaces, parametrische Kurven, NURBS und Partikel [Aut18]. Des Weiteren kann das Dateiformat Transformationen, Kameras, Lights und Materials enthalten [Aut18]. Materialrepräsentationen (und dadurch auch Lichtrepräsentationen) werden in dem Dateiformat jedoch nicht formalisiert. Alembic gibt keine bestimmten Shader-Modelle vor und ist laut eigener Aussage nicht für die Übertragung von Shadern sondern von anwendungsspezifischen Materialinformationen designt [Son]. Das Erstellen eines Materials funktioniert durch das Zuweisen eines Ziel-Renderers, z.B. „maya“ oder „arnold“, und einem eigens angegeben Shadertypen bzw. den dazugehörigen Shadereigenschaften [Son].

4.5 COLLADA

Dieses Dateiformat verwendet eine XML-basierte Beschreibung für die Assets. Dadurch ist das Auslesen von Dateien wie bei allen Textformaten im Vergleich zu binären Formaten langsamer [Lin17]. Es wurde Jahre vor glTF unter der Khronos Group veröffentlicht. COLLADA unterstützt eine große Bandbreite an Daten, die für Echtzeitrenderer relevant sind, unter anderem Hierarchien, Transformationen, Meshes, Kurven, Materialien und Texturen, Animationen von unterschiedlichen

Objekten und Parametern mit Keyframes oder gebackenen Animationesergebnissen, Kameras und Lichter [BFS08]. Generell ist die Spezifikation von manchen Daten, z.B. für Animationsdefinitionen wie Blend Shapes, unklar, was es schwer macht einen konkreten Exporter zu schreiben [Lin17].

Im Jahr 2017 publizierte die Khronos Group in ihrem Statement zu COLLADA: „The complexity of 3-D makes meaningful standardization difficult to do, enforce, and have carried out.“ [Khr17]. Auch in ihrem DOM-Wiki schreiben sie selbst, dass der Import und Export Support für COLLADA limitiert sei und daran scheitere, einen Bruchteil des Dateiformates zu realisieren [Khr20].

4.6 USD

Pixars USD ist ein OpenSource-Format für den 3D-Assetaustausch. Es hat eine binäre und eine ASCII-Repräsentation und wird in Layern aufgebaut. Das bedeutet, dass unterschiedliche Assets in verschiedenen USD-Dateien bearbeitet werden können, die dann durch Referenzierung in eine USD-Szene (genannt Stage) zusammengestellt werden. Die Aufteilung der 3D-Assets in verschiedene Dateien bzw. Layers fördert eine konfliktfreie Bearbeitung der Daten von mehreren Entwickler:innen [Pix21a].

USD beinhaltet noch weitere Konzepte: verschiedene Varianten von Attributen eines Assets können gespeichert werden, es gibt neben normalen Referenzierungen auch sogenannte Payloads, die man optional in die Szene laden und wieder entladen kann, um den Speicher effizienter zu managen, und Vererbung (Inherits) von Attributen anderer Objekte [Sid].

Eine Vielzahl an Assets wird unterstützt. Neben verschiedenen Geometrien wie Meshes, NURBS, Kurven und einige primitive Formen wie Kugeln oder Würfel kann USD auch Transformationen, Shader Graphs bzw. Materials, Lights, Linear Blend Skinning, Blend Shapes und Rigid Body Physics beinhalten [Pix21a; Pix21b].

4.7 Datasmith

Datasmith ist in erster Linie ein Toolset, um Assets aus verschiedenen Programmen in Unreal Engine zu importieren. Es unterstützt den Import von Daten aus bestimmten CAD- und DCC-Programmen, z.B. Rhino, 3ds Max, VRED etc. [Epi22d]. Für manche Programme bietet Datasmith auch ein Export-Plugin an, mit dem man eine udatasmith-Datei exportiert. Für VRED gibt es ein Datasmith-Export-Plugin, das eine FBX-Dtaei exportiert [Epi22d]. In Unreal Engine können mit den Datasmith-Import-Plugins entweder native CAD-Dateien, FBX-Dateien, oder udatasmith-Dateien importiert werden.

Mit Datasmith werden Szenenhierarchien, Geometrien und ihre Transformationen, Materialien, Texturen, Lichter, Kameras und gebackene Animationen übertragen. Unter Animationen fallen hier jedoch nur Animationen, die die Transformation von ganzen Objekten betreffen [Epi22d; Epi22e]. Deformationen oder Animationen mit Rigs werden nicht unterstützt [Epi22e].

Datasmith ist also kein klassisches intermediäres Dateiformat wie die bisherigen oben genannten Formate, sondern eine Sammlung an verschiedenen Austauschmöglichkeiten, um Assets anderer 3D-Anwendungen in eine von Unreal Engine unterstützte Datenstruktur zu bringen.

5 NVIDIA Omniverse Ansatz zur Workflowgestaltung

NVIDIA ist ein bekannter Grafikkartenhersteller, der bereits eigene 3D-Standards verwirklichte, z.B. NVIDIA Material Definition Language (MDL). Sie veröffentlichten NVIDIA Omniverse als eine freie, erweiterbare Plattform für die virtuelle Kollaboration von 3D-Entwickler:innen [NVI22k]. Die Plattform ist eine Sammlung von verschiedenen Anwendungen und Diensten, die zur Realisierung von 3D-Echtzeitvisualisierungen dienen können. NVIDIA spricht damit verschiedene Branchen an, darunter Game Entwicklung, Architecture, Engineering and Construction (AEC)-Visualisierung, wissenschaftliche Visualisierung und die Medien- und Entertainment-Branche [NVI22k].

NVIDIA Omniverse ist eine Art Marktplatz, Hub oder Launcher, um Services zu installieren und zu managen. Um den Launcher zu benutzen, ist ein kostenloser Benutzeraccount bei NVIDIA notwendig. Das Programm beinhaltet eine Übersicht der Apps und Services, die man bei Bedarf herunterladen kann, eine Verwaltung der installierten Tools, sowie eine Oberfläche für die Einrichtung eines Dateiservers und ein Lernbereich mit Tutorials für Omniverse.

5.1 Schlüsselkonzepte von NVIDIA Omniverse

Laut der NVIDIA Omniverse Dokumentation besteht die Plattform aus fünf Schlüsselservices (siehe Abbildung 5.1). Das Kernstück für die Kollaborations- und Workflowgestaltung von Omniverse ist ein Dateiserver namens Nucleus, der es den Nutzer:innen ermöglicht auf 3D-Assets über ihr Netzwerk zuzugreifen. Es können mehrere Clients gleichzeitig durch verschiedene Anwendungen an einem Projekt arbeiten. Manche Clientverbindungen ermöglichen eine Live-Synchronisation mit dem Nucleus Server. Die Zusammenarbeit folgt einem Publisher/Subscriber-Modell, bei dem ein Client als Publisher Modifizierungen an Assets in die Nucleus-Cloud veröffentlichen kann und als Subscriber Änderungen anderer Clients an Assets erhalten kann [NVI22g]. Die Szenen können durch „Checkpoints“ versioniert werden. Allerdings passiert das während eines Live-Syncs nicht automatisch, sondern Checkpoints müssen explizit vor oder nach einem Live-Sync erstellt werden [NVI22b].

NVIDIA hat sich das grundlegende Layer-Konzept von Pixars USD von Nutzen gemacht und hat USD als Basisformat implementiert. Wobei sie das Format mit ihrem eigenen MDL-Schema für Materialien und ihrer PhysX Bibliothek für physikalisch-basierte Simulationen erweitert haben [NVI22h; NVI22i].

Für den Zugriff auf den Nucleus-Server wurden für einige bestehende 3D-Applikationen sogenannte Connectors geschrieben. Das sind Plugins, die über den Launcher installiert werden können und danach in den 3D-Anwendungen zu Verfügung stehen. Solche Connectors existieren für Maya, Revit, 3DS Max, Rhino, einige Versionen von Unreal Engine 4 und weitere 3D-Applikationen. Allerdings

Abbildung 5.1: Fünf Schlüsselservices von NVIDIA Omniverse [NVI22j]

ist die Aufzählung an Logos in Abbildung 5.1 unter Connect irreführend, denn beispielsweise für Unity existiert derzeit kein offizieller Connector. Allerdings kann man über eine C++ Omniverse Connect API eigene Connectors schreiben wie in Abschnitt 5.2 beschrieben.

Einige Connect-Verbindungen zu Nucleus sind bidirektional, d. h. Änderungen an einer Szene können auf den Nucleus-Server gepushed werden und die Assets können mit den Änderungen anderer Clients synchronisiert und dargestellt werden. Andere Verbindungen sind wiederum unidirektional, was bedeutet, dass sie lediglich Assets auf dem Server veröffentlichen können, aber keine Informationen oder Asset-Änderungen von anderen Parteien erhalten [NVI22c].

NVIDIA bietet eine Vielzahl an eigenen 3D-Anwendungen, die auf der Omniverse-Plattform aufbauen und für die Erstellung und das Rendern von 3D-Simulationen unterschiedlicher Art verwendet werden können, z.B.:

- Audio2Face: Lip Sync Simulationen
- Isaac Sim: Robotics Simulationen
- Kaolin App: 3D Deep Learning Anwendung
- Machinima: Animiertes Story Telling
- Create: 3D-Editor
- View: AEC-Visualisierungen
- etc.

Das Omniverse Kit kann zum Erstellen eigener Omniverse Apps verwendet werden. Es gibt dafür bereits Bauteile bzw. Extensions, die zum Aufbau von Omniverse Apps mit dem Kit hilfreich sein können [NVI22f]. Ein großer Nachteil der Render-Applikationen von Omniverse ist, dass sie ausschließlich mit einer NVIDIA RTX-Grafikkarte laufen. Das limitiert das Publikum auf den Kreis der NVIDIA RTX-Grafikkarten-Besitzer:innen.

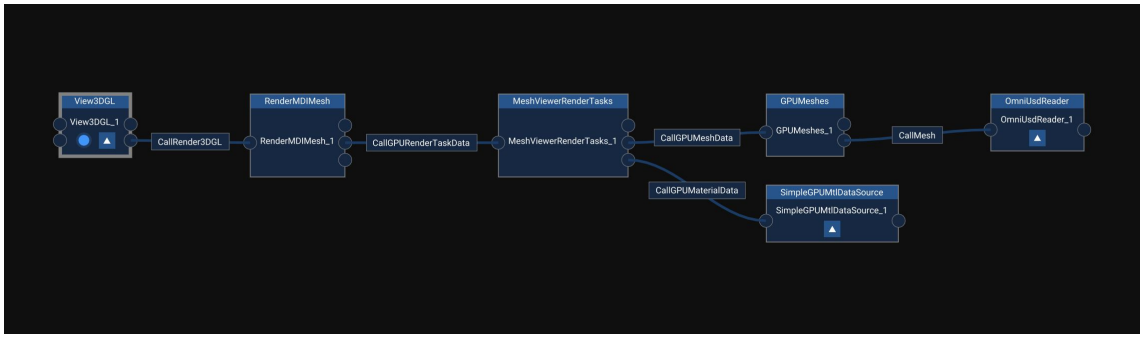

Abbildung 5.2: MegaMol-Kofigurationsgraph für simples Meshrendering

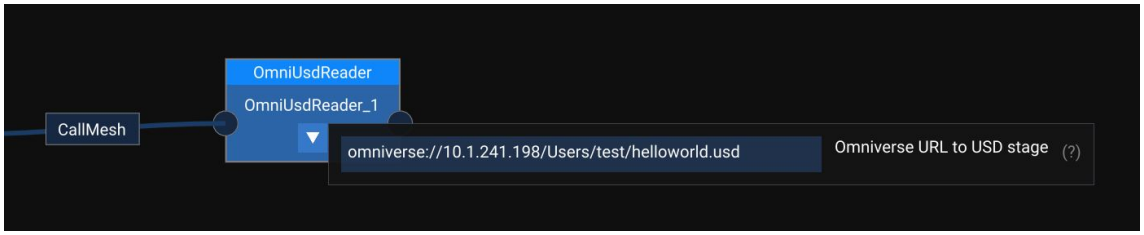

Abbildung 5.3: Omiverse Connector als MegaMol-Modul „OmniUsdReader“

5.2 Implementierung eines Omniverse Connector Prototypen

Im Rahmen dieser Arbeit wurde ein Omniverse Connector Prototyp für das Visualisierungsframework MegaMol geschrieben, das vom Visualisierungsinstitut der Universität Stuttgart entwickelt wurde [Vis].

MegaMol-Projekte werden mithilfe eines Konfigurationsgraphen aufgebaut, der aus unterschiedlichen Modulen besteht (siehe Abbildung 5.2). Module können unterschiedlicher Art sein und werden durch Callbeziehungen miteinander verbunden. Ein aufrufendes Modul weiter links im Graphen wird Caller genannt, das aufgerufene Modul weiter rechts im Graphen ist ein Callee. Der MegaMol-Connector wurde in Form eines Mesh-Moduls umgesetzt, der eine USD-Stage über eine Omniverse-URL als String-Parameter einliest (siehe Abbildung 5.3). Nach einer erfolgreichen Verbindung zum Nucleusserver und der vorhandenen USD-Datei liest der Connector Meshinformationen der Stage aus, verarbeitet sie und übergibt sie an den Caller des Moduls.

Der erste Schritt der Entwicklung des Connectors ist, über den Omniverse Launcher das Connect Sample herunterzuladen. Das Connect Sample ist eine Ordnerstruktur, die verschiedene Skripte und Symlinks zu den Omniverse Client- und USD-Bibliotheken umfasst. Es enthält zwei C++-Codebeispiele, um Konsolenapplikation zu bauen, die sich mit einem Nucleusserver verbinden. Mit einem dieser Codebeispiele erstellt man eine Applikation, die durch eine angegebene USD-Stage iteriert und alle „Primpfade“ ausgibt. Ein Prim ist ein Objekt in einer USD-Szene und kann von unterschiedlichem Typ sein, z.B. ein Mesh, ein Shader, eine Xform (Transformation) etc. Es hat einen zugehörigen Pfad in der USD-Hierarchie, und bestimmte typabhängige Attribute und „Primvars“ [Pix21a]. Primvars sind benutzerdefinierte Attribute eines Prims. Mit der zweiten Konsolenapplikation erhält man Live-Benachrichtigungen über Änderungen einer getrackten

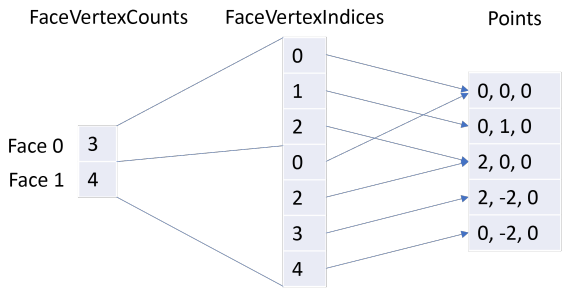

Abbildung 5.4: Indizierung der Points eines USD-Meshes mit zwei Faces und variierender Anzahl an FaceVertices

USD-Datei und schreibt die aktuelle USD-Stage in eine lokale USDA-Datei (Textversion des USD-Formates). Zusätzlich enthält das Connect Sample ein Programm, das eine simple USD-Szene auf dem Nucleusserver erstellt.

Um die lokalen Abhängigkeiten, die für den Connector benötigt werden, zu referenzieren, muss beim Bauen von MegaMol mit dem Omniverse Connector aktuell der Pfad zur OmniClient-Library und zu den USD-Libraries angegeben werden. Die Abhängigkeiten sind bereits gebaute Binärdateien, die über Omniverse heruntergeladen werden. USD benutzt eine ältere Version von Intels Thread Building Blocks (TBB), die nicht kompatibel mit einer bereits in MegaMol integrierten neueren TBB-Version ist. MegaMol-Plugins die TBB bereits verwenden, können also nicht gleichzeitig gebaut werden.

Das Testen der Connect Samples und die Entwicklung des MegaMol-Connectors sowie die Verwendung der Omniverse-Suite wurde mit folgendem Setup durchgeführt: Windows 10, NVIDIA GeForce RTX 2080 Ti GPU, Intel Core i9-9900K CPU sowie 32 GB RAM. Der Nucleusserver wurde von einer VM im Netzwerk gehostet. Die Programmierumgebung war Visual Studio 2019 mit C++ 2017 als Sprachstandard.

Sobald eine Omniverse URL dem OmniUsdReader-Modul als Stringparameter übergeben wird, versucht es, sich zu dem Nucleusserver zu verbinden und die USD-Stage zu traversieren. Der MegaMol-Connector betrachtet ausschließlich Meshes, alle anderen Prims werden aktuell ignoriert. Von einem Mesh werden folgende Attribute abgefragt: die lokale Transformationsmatrix, FaceVertexCounts, FaceVertexIndices, Points, Normalen und Texturkoordinaten, die in USD als Primvar gespeichert werden [Pixa]. Anschließend wird die lokale Transformationsmatrix auf die Pointsliste des Meshes angewendet und seine Normalen werden gemäß rotiert.

FaceVertexCounts ist eine USD-Datenstruktur, die auflistet, wie viele Vertices eine Face beschreiben. Die Anzahl an Vertices kann für ein USD-Mesh pro Face also variieren. Diese Liste wird in Kombination mit den FaceVertexIndices verwendet, um die Punkte zu indizieren (siehe Abbildung 5.4). Der implementierte OmniUsdReader kann ausschließlich Meshes darstellen, die aus Dreiecken und Vierecken bestehen, wobei Vierecke modulintern in zwei Dreiecke unterteilt werden.

Da USD-Meshes verschiedene Normaleninterpolationsmodi haben können [Pixb], ist auch eine Fallunterscheidung im Code eingebaut. Es wird in dem Connector nur zwischen „vertex“ und „faceVarying“ unterschieden, was bedeutet, dass die Anzahl der Normalen entweder der Anzahl

an Points („vertex“) oder der Anzahl an Face-Vertex-Punkten („faceVarying“) entspricht. Da die MeshDataAccessCollection in MegaMol, die zuständig für die Bereitstellung der Meshinformationen ist, nur eine Indexliste für alle Vertexattribute speichert, wird in dem Omniverse-Modul für eine faceVarying-Normaleninterpolation die Positionsliste komplett ausgerollt. Es gibt noch weitere Interpolationsmodi, die in dem Connector aber bisher ignoriert werden, z.B „varying“ [Pixb]. Jedes Vertexattribut kann in USD einen anderen Interpolationsmodus verwenden.

Das OmniUsdReader-Modul berechnet also Positionen, Normalen und Texturkoordinaten, eine Indexliste und eine BoundingBox und stellt sie dem Caller bereit. Getestet wurde der Connector hauptsächlich mit Meshes, die in der Omniverse Create-Applikation erstellt wurden. Eine Beispiel-Szene aus der Omniverse Create App ist in Abbildung 5.5 oben und das entsprechende Ergebnis in MegaMol in Abbildung 5.5 unten zu sehen.

Aktuell sind einige Defizite des MegaMol-Moduls bekannt. Eine Verkettung von mehreren Mesh-Modulen ist zum Beispiel nicht möglich. Beim Testen von komplexeren Szenen, die ursprünglich aus Unreal Engine oder Rhino kommen, gibt es noch nicht identifizierte Probleme mit der Darstellung und Position einiger Meshes.

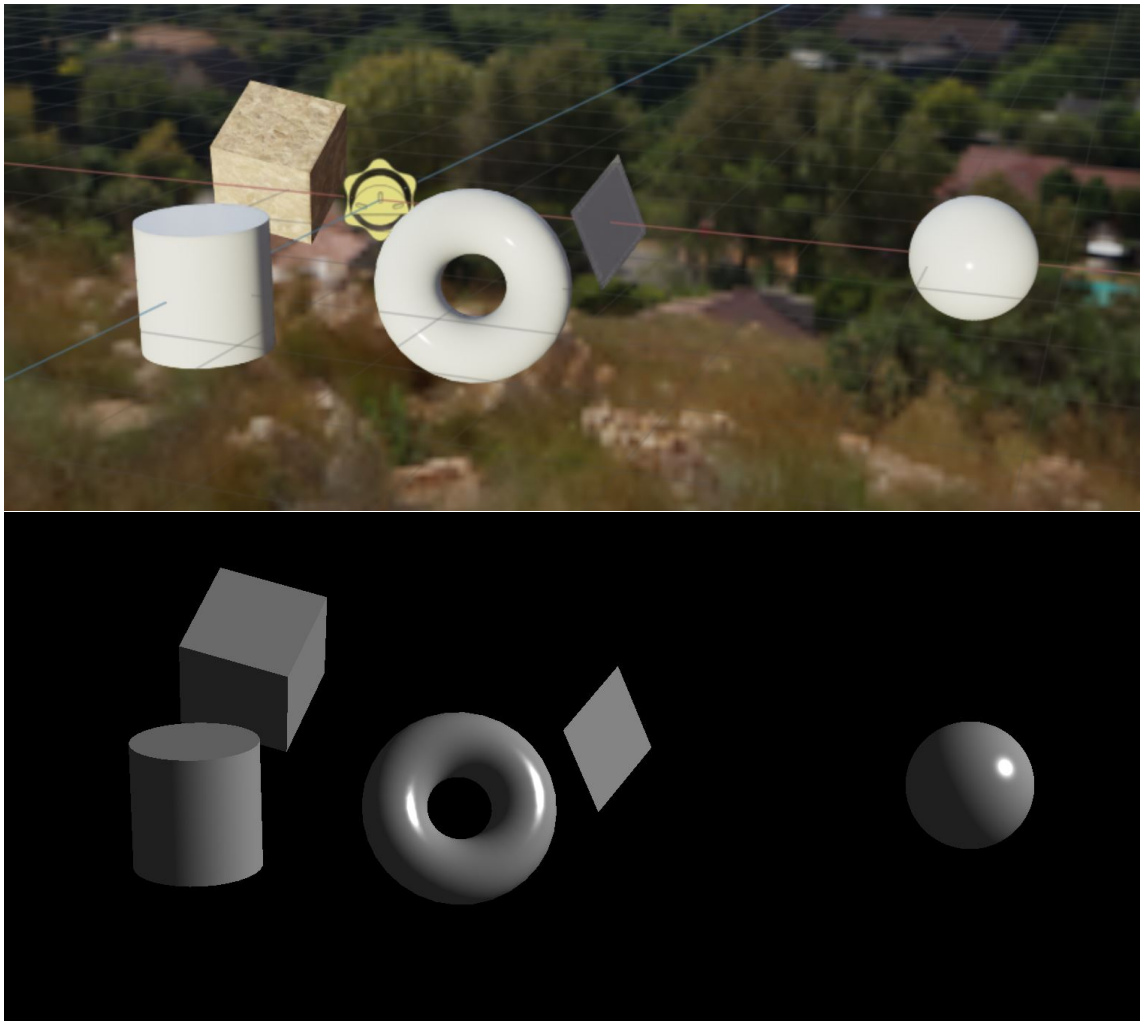

Abbildung 5.5: Testszene in der Omniverse Create Anwendung (oben) und übertragene Testszene in MegaMol (unten)

6 Experteninterviews zu Workflowstandards und Nutzeranforderungen

Aufgrund des Literaturmangels zu konkreten Problemen von Produktionspipelines, wurden halbstrukturierte Experteninterviews durchgeführt. Zwölf Expert:innen wurden einzeln anhand eines vorab formulierten Leitfadens zu ihrer Assetpipeline, sowie zu Erfahrungen mit Import- und Exportprozessen und Softwarenutzungsmustern befragt. Die Befragten decken unterschiedliche Teile der Pipeline ab, hatten in der Vergangenheit an unterschiedlichen Projekten gearbeitet und arbeiten teilweise in unterschiedlichen Branchen. Die einzelnen Interviews haben daher verschiedene Ausprägungen in der Intensität einzelner Themenbereiche angenommen.

6.1 Profil der Befragten

Die Auswahl der Interviewteilnehmer:innen entwickelte sich aus dem beruflichen Umfeld der Autorin und der Bertreuer dieser Bachelorarbeit. Die Befragten sind in der Entwicklung von 3D-Inhalten tätig. Ein Teil der Interviews wurde mit Architekt:innen und Entwickler:innen geführt mit Projekten wie Architekturvisualisierungen oder -simulationen. Einige Befragte sind in der interaktiven Medienbranche einzuordnen mit Projekten wie Produktvisualisierungen/ -simulationen, Animationen, Virtual Reality (VR)-Anwendungen, Virtual Production, Videospiele etc.

In Kategorien anhand der üblichen Pipelinestichworte decken die Befragten folgende ab: 3D-Modellierung, Texturierung, (Technical) 3D-Animation, 3D-Softwaredevelopment und Pipeline TD. Die einzelnen Personen lassen sich nicht ausschließlich einer Kategorie zuordnen, sondern sind meist in mehreren Bereichen unterschiedlich erfahren. Manche begleiten oder führen teilweise sogar alle Schritte aus. Auch der Bezug zu Echtzeitrenderern ist sehr unterschiedlich unter den Interviewten. Einige arbeiten hauptsächlich mit Offline-Rendering-Programmen und hatten ein paar Erfahrungen mit dem Export in Echtzeitengines gemacht. Andere wiederum arbeiten fast ausschließlich in Echtzeitengines und haben seltener 3D-Modelle in DCC- oder CAD-Anwendungen erstellt und aufbereitet. Weitere haben in der gesamten Pipeline tiefgreifendere Kenntnisse.

6.2 Leitfaden des Interviews

Vor den Interviews wurde ein Fragenkatalog formuliert, der nicht immer gleichermaßen stringent befolgt wurde. Je nach Gebiet des/der Teilnehmer:in wurden Fragen umformuliert, weggelassen oder hinzugefügt. Eine Abstraktion des Fragenkatalogs zu folgendem Leitfaden ist in folgender Auflistung zu sehen:

1. Erfragen der Tätigkeit, der bisherigen Projekte, Erfahrungen mit Programmen, Nutzungsmuster, Dateiformaten und bestehendem Workflow
2. Detailliertere Fragen zu Erfahrungen mit dem Austausch von 3D-Objekten, Materialien, Lichtquellen, Animationen, Rigs und Simulationen zwischen verschiedenen Programmen
3. Fragen nach Empfinden des aktuellen Workflows, freie Redezeit für positive und negative Aspekte und Wünsche an bisherigen Workflows

6.3 Nachbereitung der Interviews

Die Interviews wurden aufgenommen und Antworten fragmentarisch transkribiert. Gleichzeitig wurden die Fragmente verschiedenen Themenbereichen zugeordnet, um Zusammenhänge zwischen den Erfahrungen der Teilnehmer:innen zu ziehen. Es ließen sich sehr unterschiedliche Contentpipelines bzw. Pipelinefragmente extrahieren. Im Folgenden werden einige exemplarisch ausgewählte Pipelines beschrieben und Nutzungsmuster zusammengefasst.

6.4 Beispiele für Contentpipelines

In Abbildung 6.1 bis Abbildung 6.4 sind Asset-Pipelines abgebildet, um im Folgenden Gemeinsamkeiten und Unterschiede zwischen verschiedenen Nutzungsmustern auszuarbeiten. Diese Beispiele sind abstrahiert und bilden quasi einen Querschnitt der Austauschprozesse der Befragten ab.

Abbildung 6.1 zeigt eine Pipeline für ein Projekt, das auf CAD-Daten basiert und keine Level of Details (LODs) oder Animationen im Austausch beinhaltet. Das CAD-Modell liegt in einem beliebigen CAD-Format vor, das in einer DCC-Anwendung geöffnet werden kann. Es wird dort zu einem Mesh konvertiert und aufkommende Fehler werden manuell behoben. Da das Mesh nach der Tessellierung zu viele Polygone oder Deformationen hat, wird die Topologie optimiert. Danach werden die verschiedenen Objekte in der gleichen Anwendung in Materialiengruppen unterteilt. Dann wird das Mesh mit der Materialunterteilung in ein intermediäres Dateiformat exportiert und in die Game Engine importiert. Die finalen Materialien werden allerdings erst in der Game Engine erstellt und den vorher eingeordneten Materialgruppen zugewiesen. Lighting wird erst in der Game Engine vorgenommen und, wenn das Projekt das verlangt, können dort noch kleinere Animationen kreiert oder LODs erstellt werden. Außerdem muss überprüft werden, falls Gameskripte bestimmten Assets zugewiesen wurden oder ein Skript bestimmte Assets referenziert, ob diese Zuweisungen bzw. Referenzierung korrekt ist.

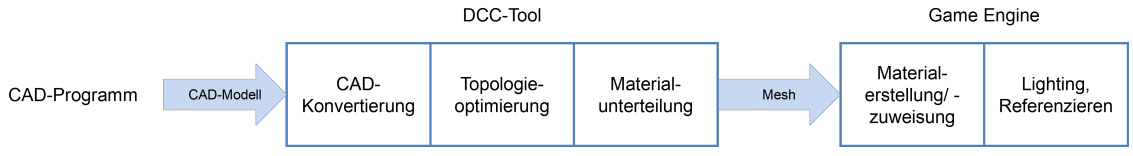

Abbildung 6.1: Beispiel für eine Assetpipeline mit CAD-Objekten, Materialunterteilung und Materialmapping

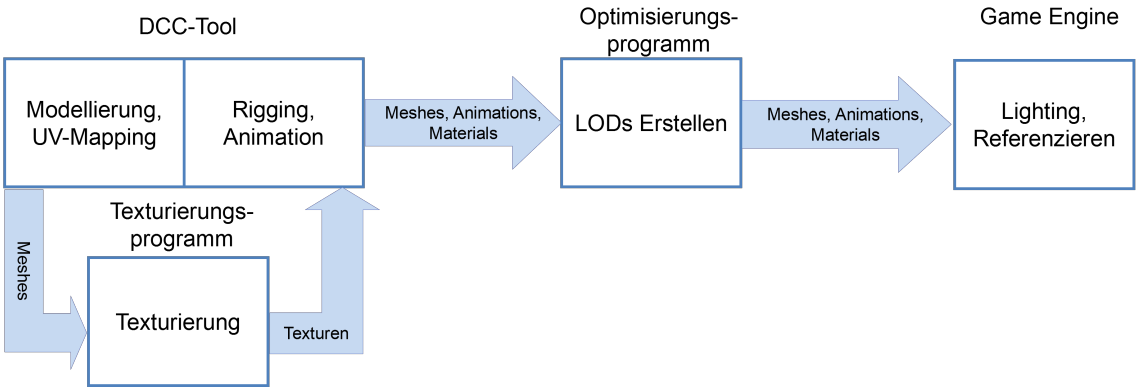

Abbildung 6.2: Beispiel für eine Assetpipeline mit Texturierung, LODs, modellierten Objekten, Animationen, Rigs

Die zweite Content-Pipeline in Abbildung 6.2 zeigt ein Projekt, in dem im Gegensatz zu der ersten Pipeline die Meshes in einer DCC-Anwendung modelliert und UVs hinzugefügt werden, um später texturiert zu werden. In der gleichen Anwendung wird daraufhin ein Rig erstellt und animiert. Parallel dazu werden die Meshes mit UVs exportiert und in eine Anwendung importiert, in dem ein Artist verschiedene Texturen erstellen kann. Anschließend kommen die Assets in ein Optimisierungsprogramm, in dem LODs von den Assets erstellt werden. Von da aus werden alle Assets in die Game Engine gebracht, in der das Lighting geschieht und die korrekte Referenzierung von Skripten und 3D-Assets überprüft oder neu gesetzt wird.

Die Workflows in Abbildung 6.3 und Abbildung 6.4 sind besonders. Man importiert oder exportiert keine Datei explizit. Der Austausch der Daten erfolgt implizit durch ein Plugin, das eine Anwendung unmittelbar in die andere integriert. In Abbildung 6.3 sieht man eine CAD-Anwendung, die in einer Echtzeitengine eingebunden ist. Konkret wurde in diesem Fall Rhino Inside Unity verwendet, ein Plugin für Unity. Es wurden nur die in Rhino bzw. Grasshopper tessellierten Geometrien ausgetauscht. In Abbildung 6.4 wurde ein Echtzeitrenderer innerhalb eines CAD-Programms verwendet. In diesem Fall wurde Enscape verwendet, ein echtzeitfähiger Renderer, der als Plugin in Rhino für Architekturvisualisierung verwendet wird. Man erstellt hier nicht explizit Meshes, die Integration der Geometrie erfolgt automatisch. Allerdings werden nicht alle Rhinomaterialien unterstützt, daher muss man, wenn man Materialien möchte, im Enscape-Plugin zusätzliche erstellen.

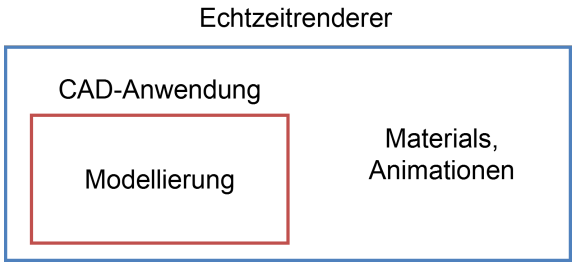

Abbildung 6.3: Beispiel für einen Workflow mit CAD-Plugin, z.B. Rhino Inside

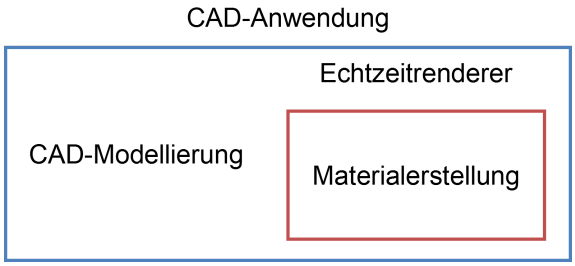

Abbildung 6.4: Beispiel für einen Workflow mit Echtzeitrendererererweiterung, z.B. Enscape

6.5 Austauschprozesse und Workflows

Im Folgenden werden verschiedene Aspekte von Content-Pipelines auf der Grundlage der beschriebenen Workflows der Interviewteilnehmer:innen zusammengefasst.

6.5.1 Datenkomplexität und Auswahl der Anwendungen

Pipelineprobleme häufen sich mit der Komplextität, Größe oder Relevanz der Daten. Die in Abbildung 6.1 und Abbildung 6.2 dargestellten Beispiele sind beliebig kürzbar oder erweiterbar. Wenn es keine detaillierten Meshes gibt, braucht man keine LODs. Wenn man keine Texturen benutzt, muss man nicht in ein Texturierungsprogramm usw. Häufig antworteten die Befragten auf viele Fragen mit „Das ist von dem Projekt abhängig.“, „Das ist vom Mesh abhängig.“ oder „Das hängt davon ab, wie wichtig das Asset ist.“.

Game Engines bieten viele Features selbst an, z.B. Animationseditoren oder simple Geometrieoperationen, und auch schon vorgefertigte Assets, beispielsweise Materials. Das genügt für simple Szenen, die Werkzeuge sind allerdings nicht in dem Detailgrad implementiert wie in anderen DCC-Tools. Ob ein Asset in einer anderen Anwendung oder in der Echtzeitengine angefertigt bzw. verarbeitet wird, hängt von den Künstler:innen ab und von den Instrumenten, die einem die Echtzeitengine oder 3D-Anwendung bereits bietet.

Neben Features auf Werkzeugebene kann die Auswahl oder Vorgabe eines Programmes den reibungslosen Datenfluss stören, wenn es keine gängigen Dateiformate exportieren oder importieren kann. Eventuell müssen die Daten dann auf Umwegen passend konvertiert werden.

6.5.2 Schnittstellen zwischen Programmen

Die Ausführung der Schnittstellen zwischen Programmen kann unterschiedlich sein:

1. Der Export und Import der Assets geschieht durch ein intermediäres Dateiformat wie in den meisten Fällen.
2. Das Programm ermöglicht, Assets direkt in ein passendes Format für den Echtzeitrenderer abzuspeichern. In Texturierungsprogrammen wie Substance Painter oder Quixel Mixer gibt man beispielsweise die Zielplattform beim Exportieren an.

3. Der Echtzeitrenderer unterstützt das Format der vorherigen Anwendung, zum Beispiel durch ein Plugin. In Unreal Engine gibt es zum Beispiel den Datasmith Importer, der native CAD-Daten z.B. aus Rhino, aber auch aus anderen Programmen importieren kann.
4. Ein Programm kann in ein anderes als Erweiterung der eigentlichen Anwendung wie in Abbildung 6.3 und Abbildung 6.4 integriert sein. Der Datenexport und -import findet hierbei nicht extern statt.

6.5.3 Richtung der 3D-Assets durch die Pipeline

Bei dem größten Teil der Befragten sind die Pipelines unidirektional. In wenigen Fällen gibt es Ausnahmen, beispielsweise bei der Übernahme eines bereits in einer Game Engine angefangenen Projektes. In dem Fall müsste man, um in einem DCC-Tool etwas zu verändern, das Asset anfangs einmal exportieren. Bei Interviewten, die integrierte oder synchronisierte Schnittstellen benutzen, ist Bidirektionalität manchmal implizit vorhanden.

6.5.4 Effiziente Iterationen der Assetpipeline

Da die Entwicklung eines Assets ein Designprozess ist, müssen Export und Import oft wiederholt werden. Denn man updatet immer die ursprüngliche Workfile eines Assets und nimmt normalerweise keine direkten Veränderungen mehr in der Game Engine vor. So stellt man sicher, dass alle Künstler:innen auf den gleichen Rohdaten arbeiten. Diese Iteration des Imports ist möglichst effizient zu gestalten. Der Importprozess kann nämlich an sich lange dauern und mit Instruktionen verbunden sein, um das Asset wieder richtig in das Programm einzupflegen.

In Game Engines kann es automatisierte (Re-)Importhilfen geben. In Unreal Engine gibt es beispielsweise Visual Dataprep oder man schreibt selbst ein Skript, das die gewünschten Schritte auf den Daten ausführt. Häufig wird Dataprep für den Austausch von 3D-Assets, vor allem Materials, verwendet. Mehrere Objekte werden automatisch zu einem Objekt zusammengeführt oder für die weitere Bearbeitung mit bestimmten Tags versehen. Die Auswahl der Assets, auf denen eine bestimmte Aktion ausgeführt wird, basieren größtenteils auf Regexoperationen und bauen auf einer stabilen Namensgebung, Hierarchie oder stabilen Metadaten auf.

6.5.5 Assetoptimierungen

Neben visuellen Fehlern ist häufig die schlechte Performance eines Assets ein Grund für eine nötige Nachbearbeitung und einen Reimport. In den Game Engines gibt es Statistiken, die ausgegeben und dargestellt werden können, und so zum „Profiling“ benutzt werden. Man importiert die Assets also in die Echtzeitengine und lässt sich dort bestimmte Renderingzeitwerte, z.B. Drawtime anzeigen. So kann man erarbeiten, was die Ursache für die schlechte Performance ist, denn das kann unterschiedliche Gründe haben.

Eventuell muss man die 3D-Assets dann optimieren, wenn beispielsweise die Anzahl zu rendernder Polygone zu hoch ist, müssen die Polygone reduziert werden oder auf Details verzichtet werden, die klein, unauffällig oder verdeckt sind, aber aus vielen Polygonen bestehen, beispielsweise Schrauben. Oder man setzt sie auf eine andere Weise um, z.B. mit einer Annäherung eines Details auf einer

Textur. Die Zeit eines Drawcalls oder die Menge an Drawcalls auf der CPU kann auch ein Problem sein, ebenso die Shader Komplexität, bzw. die Anzahl an Instruktionen auf der GPU. Es gibt noch viele weitere Engpässe, die man durch Profiling und Assetoptimierungen lösen kann.

6.5.6 Kollaboration

Dieses Thema wurde in den Interviews nicht intensiv bearbeitet. Es gibt CAD-Anwendungen, die eine synchrone Cloudzusammenarbeit ermöglichen, beispielsweise Revits Autodesk BIM 360. Auch für Unity und Unreal Engine gibt es Live-Share-Lösungen, sie sind aber nicht populär unter den Befragten.

Ebenso ist die Versionierung ein Thema, das sicherlich zu großen Pipelineproblemen führen kann. Einige versionieren ihre 3D-Assets mit Git oder Perforce.

6.6 Austauscherfahrungen mit 3D-Assets

In den folgenden Abschnitten werden Erfahrungen bei der Überführung von Assets geschildert, von denen die Befragten berichteten. Der Abschnitt dient dazu herauszufinden, wie Assets für gewöhnlich übernommen werden und welche Optimierungen und Schritte man häufig vornehmen muss, um Assets korrekt in ein Echtzeitprojekt einzupflegen.

6.6.1 Koordinatensystem

Die Up-Axis variiert von Programm zu Programm zwischen Z-Achse und Y-Achse. Manche intermediären Dateiformate können die Information speichern, damit das nächste Programm das Objekt gegebenenfalls direkt rotieren kann. Auch die Einheiten können ein Problem sein. Das lässt sich zwar schnell durch Reskalieren lösen, es kann allerdings Probleme geben, sobald eine Animation für das Objekt existiert und die Bewegungen nicht mehr zum neuskalierten Objekt passen. In DCC-Anwendungen kann man die Einheit normalerweise einstellen.

6.6.2 Hierarchien

Meshes können unterschiedliche Aufteilungen haben. In der AEC-Branche sortiert man häufig nach Bauteilen, in der Rendering-Branche möchte man die 3D-Objekte gelegentlich nach anderen Kriterien sortieren, z.B. nach Material. Manchmal wird die Übernahme einer Hierarchie sogar bewusst ignoriert.

DCC- oder CAD-Anwendungen, die ohne herkömmliche Hierarchien arbeiten, sollten ihre interne Struktur bei einem Exportvorgang in ein intermediäres Dateiformat, das Hierarchien speichert, übersetzen. Ein Beispiel dafür ist der Export von Rhino3d nach Unreal Engine mit Datasmith. Datasmith interpretiert Layer und Blöcke aus Rhino in eine Szenenhierarchie.

6.6.3 Geometrie

Es gibt verschiedene Szenarien, wie die Daten akquiriert werden. Entweder ein Asset wird extra modelliert für die Echtzeitengine wie in Abbildung 6.2, wobei von Anfang an auf die Topologie und Kompatibilität mit dem finalen Renderer geachtet wird. Die Interviewten benutzen dafür Programme wie Blender, Maya oder ZBrush. Man kann sich Assets aus verschiedenen Assetbibliotheken holen, die im besten Fall schon für Online-Rendering gebaut sind. In Projekten, die auf Daten aus der AEC-Branche aufbauen, müssen die Entwickler:innen ein Modell normalerweise zuerst in ein Mesh konvertieren, um es in der Echtzeitengine zu rendern wie auch in Abbildung 6.1. Dabei kommen die Daten in unterschiedlichsten CAD-Formaten (IFC, 3dm, DWG, STEP, etc.). Die Tesselierung zu einem triangulierten Mesh kann ein sehr zeitintensiver Schritt in der Assetpipeline sein. Manchmal wird das Modell sogar als Mesh nachmodelliert.

Tesselierung

Die Einstellungsmöglichkeiten des Tesselierungsvorgangs können unterschiedlich zwischen Programmen sein. Viele DCC-Tools haben einen nahtlosen Übergang zwischen polygonaler Darstellung und der Darstellung von NURBS. Die Konvertierung in ein trianguliertes Mesh kann durch ein Feature in einem Programm, beim Exportieren oder beim Importieren erfolgen.

Abhängig vom Tesselierungsalgorithmus und den grundlegenden Daten ergibt die Konvertierung mehr oder weniger wünschenswerte Resultate. Topologiedeformationen entstehen nicht selten, wie z.B. Überlappungen oder Lücken zwischen zwei angrenzenden Flächen. Häufig werden flache Oberflächen auch in viele kleine Dreiecke unterteilt, was die Polygonanzahl unnötig erhöht (siehe Pipeline in Abbildung 6.1). Ein weiteres Problem kann die falsche Ausrichtung der Vertexnormalen von verschiedenen Polygonen sein, was in einer Game Engine später zu durchstichtigen Faces oder Shadingartefakten führt.

In den Interviews wurde von guten Erfahrungen mit Tesselierern berichtet, dessen Einstellungen man lokal von Patch zu Patch verändern kann. In VRED, ein Programm zum Visualisieren von Automobildesign, kann man ausgewählte Objekte einzeln auf Grundlage der hinterlegten CAD-Daten mit speziell für das Patch eingestellten Konfigurationen tesselieren. So kann man ein paar der oben genannten Artefakte verringern, da sie an den Detailgrad des zu bearbeitenden Teils angepasst werden.

Zusammenführen von Vertices

Um die Anzahl an Vertices klein zu halten und um Tesselierartefakte wie Spalten an den Rändern von Meshes zu korrigieren, kann man Vertices, die unmittelbar oder fast aufeinanderliegen, zusammenführen. So wie für die Tesselierung kann sich ein globales Zusammenführen der Vertices nach Distanz schwierig gestalten. Je nach Feinheit eines Objektes sind die Distanzen zwischen den Punkten unterschiedlich, so riskiert man Details im Mesh zu verlieren.

Reduzieren von Polygonen

Polygone kann man manuell reduzieren. Häufig bieten 3D-Programme dafür aber auch Algorithmen, die auf die Meshes angewendet werden können. Einige Anwendungen sind darauf spezialisiert, z.B. InstaLOD oder PiXYZ. Diese Programme werden in erster Linie zum Erstellen von LODs benutzt wie in der Content-Pipeline in Abbildung 6.2.

6.6.4 Lights

Viele Interviewte gaben an, dass ausschließlich in der Echtzeitengine gelightet wird, da Lighting so unterschiedlich behandelt wird in verschiedenen Renderern. Der einzige Anwendungsfall, in dem die Übertragung von Lighting relevant ist, ist die Positionsübernahme von Lichtquellen.

6.6.5 Materials

Die Materialübernahme funktioniert bei den Befragten fast nur auf einer referenziellen Ebene, d.h. es wird erkannt, welches Objekt welches Material hat. Größtenteils werden die Materials in der Game Engine aber nicht korrekt dargestellt, da die Shadingparameter sich zwischen verschiedenen Anwendungen und auch innerhalb Anwendungen zwischen verschiedenen Renderern unterscheiden. Es gibt diverse Szenarien für den Materialisierungsprozess:

1. Die finalen Materialien werden in der Game Engine erstellt, aber Materialunterteilungen werden aus der DCC-Anwendung übernommen, wie in Abbildung 6.1. Beim Import können die finalen Materialien automatisch den Objekten zugeteilt werden beispielsweise mit Dataprep Methoden (siehe Abschnitt 6.5.4). Diese Zuteilung kann nach unterschiedlichen Regeln geschehen.
    a) Die 3D-Objekte sind hierarchisch nach Materialien aufgeteilt.
    b) Die 3D-Objekte haben bestimmte Tags oder andere Metadaten, die in der Game Engine geparsed werden können.
    c) Die 3D-Objekte hatten bereits Materialien in der Exportanwendung und diese Materialien werden ausgetauscht.
2. Übernommene Materialien werden nach dem Import angepasst.

Es gibt auch Programme in denen Materialien und vor allem Texturen erstellt werden können wie Quixel Mixer oder Substance Designer. Außerdem berichteten einige von einem erfolgreichen Materialimport mit glTF.

Für AEC-Daten stellt sich die Referenzierung von Materials ein bisschen schwieriger heraus. In STEP-Dateien hat jedes Objekt ein eigenes Material. Das ist nicht optimal, denn man möchte die Materialien gerne zusammenfassen.

6.6.6 Texture Maps

Man exportiert zum Texturieren entweder ausschließlich ein Low-Poly-Mesh oder ein High-Poly-Mesh in Kombination mit einem Low-Poly-Mesh für bestimmte Bake-Prozesse aus denen Maps wie z.B. die Normal Map entstehen. Substance Painter scheint zum Erstellen von Texturen beliebt, auch genannt wurden Quixel Mixer zum Texturieren und der Renderer Marmoset Toolbag zum „Baken“ verschiedener Maps.

6.6.7 Animationen und Rigs

Viele Interviewte backen auch Animationen. Allerdings kostet das viel Speicherplatz und ist nicht besonders performant während der Laufzeit der Anwendung. Ein beliebtes intermediäres Dateiformat zum Speichern von Animationen ist FBX.

Der Grund für das Baken von Animationen ist, dass die Animation nicht gleich interpretiert wird. Darunter kann die unterschiedliche Interpolation von Keyframes in der DCC-Anwendung und in der Game Engine ein Grund sein. Echtzeitengines können auch aus Performance-Gründen Beschränkungen auf Animationen haben, beispielsweise, wenn maximal zwei Joints Einfluss auf einen Kontrollpunkt haben dürfen. Dann werden nur die beiden Joints, mit den höchsten Gewichtungen berücksichtigt und andere werden ignoriert.

Bei Rotationen in einer Animation ist ein einheitlicher Pivot-Punkt der ursprünglichen und finalen Animation wichtig. Auch die korrekte Skalierung von dem Skelett und dem Objekt sind für die korrekte Ausführung der Animation in der Game Engine relevant.

Man muss auch auf die Benennung von Joints achten. Sie sollten keine duplizierten Namen haben, sonst kann es Importprobleme in Unreal Engine geben.

6.6.8 Simulationen

Für Simulationen von Kleidung oder Partikeln wird häufig das Alembic-Dateiformat für den Transfer benutzt. Das Speicherformat ist sehr groß und kann zu Performanceproblemen führen.

Für physikalische Simulationen wurden auch benutzerdefinierte Formate erstellt, um Daten zu übertragen und einzulesen.

6.7 Diskussion des Formates

Die zwölf Interviews waren in ihrer Beschaffenheit sehr uneinheitlich. Die unterschiedlichen professionellen Gebiete ergaben eine Mischung aus Antworten auf unterschiedlichen Abstraktionsebenen und unterschiedlichen Anwendungsfällen.

Durch die veschiedenen Antworten von lediglich zwölf Teilnehmer:innen auf einen bestimmten Sachverhalt hat sich nicht immer eine Intersubjektivität erkennen lassen. Es entwickelte sich daraus keine statistische oder sachliche Studie, lediglich eine Sammlung an Arbeits- und Anwendungsmuster von zwölf Individuen aus dem 3D-Anwendungsgebiet.

Des Weiteren setzten einige Fragestellungen eine gewisse Problematik schon voraus und suggerierten somit das Dasein eines Problems. Die Tendenz eine bestimmte Antwort zu geben, war daher nicht immer unvoreingenommen.

7 Kriterienkatalog für den 3D-Austauschprozess

Der Kriterienkatalog für Austauschprozesse von Assets zwischen 3D-Anwendungen basiert auf den Workflows und den Schwierigkeiten auf Assetebene, die aus den Interviews und eigenen Erfahrungen ausgearbeitet wurden.

Der Katalog ist in Form von Fragen umgesetzt, die man an einen Austauschprozess zwischen zwei 3D-Anwendungen stellen kann. Das Ziel dieses Kriterienkatalogs ist, ihn in der Planungsphase einer Contentpipeline anzuwenden, um die Schnittstellen auf verschiedenen Detail-Ebenen zwischen Programmen zu definieren und zu testen. Das könnte dabei helfen, mögliche Informationsverluste oder Freiheitsverluste aufzuspüren und frühzeitig Lösungen oder Workarounds zu finden. Aber auch während einer Produktion kann der Katalog dabei hilfreich sein, Importfehler oder Informationsverluste nachzuvollziehen. Auf den Workflow angewendet könnte der Katalog hilfreich dabei sein, Austauschkonzepte zu definieren, und somit eine klare, einheitliche Contentpipeline für die Entwickler:innen zu schaffen.

Für viele Fragen ist eine gute Dokumentation der verwendeten Programme und Dateiformate von Vorteil. Der Kriterienkatalog sieht wie folgt aus:

1. Austausch
    a) Gibt es zwsichen den Programmen eine passende Lösung, um die zu übertragenden 3D-Assets in ein Format zu exportieren, das von dem nächsten Importierer gelesen werden kann?
    b) Welche 3D-Assets sollen ausgetauscht werden und wie werden sie in den Programmen repräsentiert?
        i. Braucht man ein intermediäres Dateiformat?
        ii. Unterstützt eines der Programme das Format des anderen?
    c) Soll bzw. kann der Austausch unidirektional oder bidirektional stattfinden?
    d) Zu welchem Zeitpunkt finden Asset-Konvertierungen statt? Gibt es Vorteile oder Nachteile bei mehreren möglichen Zeitpunkten?
    e) Ist der Zugriff von beiden Programmen auf ein Asset synchron?
2. Exporter
    a) Wie viel Kontrolle hat man über die gespeicherten Daten? Welche Einstellungen stehen zu Verfügung?
    b) Wie lange dauert der Export?

c) Welche Daten oder Datenstrukturen werden nicht exportiert, obwohl das Dateiformat sie beinhalten könnte?

3. Importer
   a) Wie ist der Import von Assets gestaltet?
      i. Wie destruktiv ist ein Reimport von aktualisierten Daten? Erkennt der Importer, z.B. anhand des Namens des Assets oder anderer Metadaten, dass ein bereits bestehendes Objekt durch eine neue Version ersetzt werden soll und erhält Eigenschaften, die in dem Importprogramm hinzugefügt wurden?
      ii. Gibt es die Möglichkeit, Schritte automatisch auszuführen um das Asset zu integrieren, z.B. für die Zuordnung von Programmskripten, die Substitution von Assets oder das Zusammenführen von Assets?
   b) Wie lange dauert der Import?
   c) Welche Daten oder Datenstrukturen werden nicht importiert, obwohl sie von dem Programm interpretiert werden könnten?
4. Dateiformat
   a) Welche wichtigen Daten und Datenstrukturen können mit dem Dateiformat nicht gespeichert werden, z.B. keine Hierarchie, keine Metadaten, keine Animationen?
   b) Enthält es redundante Daten?
   c) Wie viel Speicherplatz nimmt die Datei ein?
   d) Wird das Dateiformat während der Laufzeit der Anwendung ausgelesen? Wenn ja, wie performant ist das Auslesen der Daten?
5. Assets
   a) Szenenhierarchie und Metadaten
      i. Wie konsistent ist die Benennung von Assets?
      ii. Wie wird die Szene hierarchisiert und wie werden verschiedene Hierarchiemodelle übersetzt?
   b) Koordinatensystem
      i. Wird die Up-Axis korrekt übernommen?
      ii. Wird die Einheit korrekt übernommen?
      iii. Wie werden Transformationen umgesetzt?
   c) Oberflächen
      i. Werden Vertex-Attribute wie Normalen, Texturkoordinaten, Vertex-Farben etc. korrekt übernommen?
      ii. Ist eine Tesselierung der Oberflächen notwendig?

A. Kann man Teile des Objektes mit unterschiedlichen Einstellungen tesselieren?

B. Wie detailliert sind die Konfigurationsmöglichkeiten zum Tesselieren?

C. Müssen die Polygone reduziert werden?

D. Müssen Normalen invertiert werden?

E. Entstehen Deformationen wie Überlappungen oder Spalten zwischen Patches?

F. Können Vertices zusammengeführt werden? Wenn ja, wie differenziert geschieht das zwischen unterschiedlich detaillierten Objekten?

d) Materials

i. Auf welche Weise werden Materials übernommen?

ii. Welche Oberflächenwerte bzw. Shaderoperationen können übersetzt werden?

iii. Wie volatil ist die Material- bzw. Texturreferenzierung?

e) Rigs und Animationen

i. Wird die Animation gebacken?

ii. Welche Animationsdefinitionen können von beiden Programmen interpretiert werden?

iii. Sind Vertices auf eine Anzahl beeinflussender Joints begrenzt?

iv. Kann man die einheitliche Skalierung von Rig und Objekt sicherstellen?

v. Ist der Rotationspivot der gleiche?

vi. Ist die Interpolation zwischen Keyframes gleich?

Die Aufteilung in Austausch, Importer, Exporter, Dateiformat und Assets dient dazu, den Austauschprozess in verschiedenem Detailgrad und in verschiedenen Phasen zu betrachten. Der erste Teil definiert, wie die Schnittstelle zwischen den Programmen allgemein auf einer Workflowebene gestaltet ist. Der zweite Teil des Katalogs über Exporter ist dafür da, die Funktionen des Exportiervorgangs zu untersuchen. Danach wird auf den Importer und auf Reimportlösungen eingegangen und anschließend das intermediäre Dateiformat, insofern eins verwendet wird, analysiert. Der fünfte Part ist zum Beobachten der Entwicklung einzelner zu übertragender Assets über den gesamten Austauschprozess.

Die Voraussetzung für die Bearbeitung des Kriterienkatalog ist die Beantwortung der Frage 1.a). Wenn es keine Austauschlösung zwischen zwei Programmen gibt, müssen die Bedingungen und Mittel des Austausches verändert werden. In dem Fall muss eventuell ein weiteres Programm, das die Daten interpretieren und in ein importierbares Format bringen kann, als Zwischenschritt dienen, um den Import zu ermöglichen.

Für einen bidirektionalen Austausch sollte jede Richtung einzeln mit dem Kriterienkatalog untersucht werden. Import- und Exportprogramm tauschen im zweiten Durchgang die Rollen.

8 Evaluation von Austauschszenarien mithilfe des Kriterienkatalogs

Um zu testen, ob der Kriterienkatalog anwendbar ist, wurden zwei Assetaustauschszenarien durchgeführt. Als Probe-Szene wird eine 3dm-Rhino-Datei verwendet, die eine Hochhausstruktur als CAD-Modell abbildet, die Szene enthält viele Objekte und ist in Layern strukturiert (siehe Abbildung 8.1). Es gibt 20 Materials, die vereinzelten Objekten zugewiesen sind. Die Szene enthält keine Animationen oder Simulationen, daher wird der Animationspart des Kriterienkatalogs nicht angewendet.

Die Tests wurden mit dem Setup aus Abschnitt 5.2 durchgeführt. Die verwendeten 3D-Programme sind Rhinoceros 7 und Unreal Engine 4.27.2.

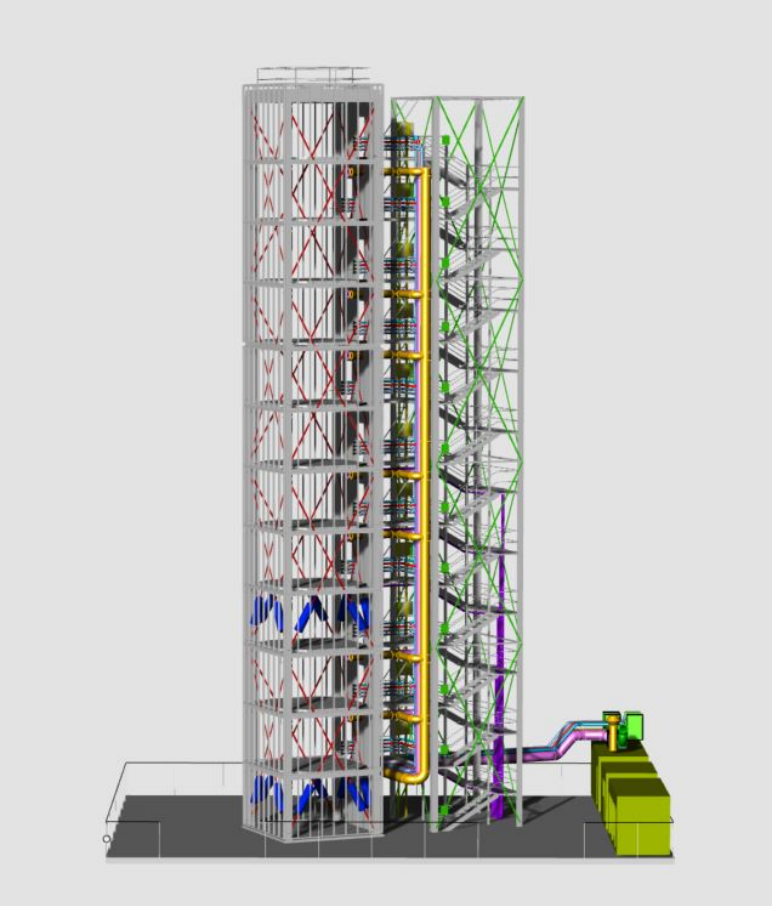

Abbildung 8.1: Testszene in Rhino, Modell des adaptiven Hochhausdemonstrators des Sonderforschungsbereichs 1244 der Universität Stuttgart [Uni]

8.1 Rhino zu Unreal Engine Export mit Datasmith

1. **Austausch**

   a) **Gibt es zwsichen den Programmen eine passende Lösung, um die zu übertragenden 3D-Assets in ein Format zu exportieren, das von dem nächsten Importierer gelesen werden kann?**

   Der Workflow mit Datasmith kann auf zwei unterschiedliche Arten stattfinden. Es gibt die Möglichkeit die Rhino-Szene als native 3dm-Datei in Unreal Engine mit Datasmith zu importieren. Die Szene wird dabei nicht in ein intermediäres Dateiformat exportiert. Mit dem zweiten Workflow wird eine udatasmith-Datei aus Rhino exportiert und in Unreal Engine als Datasmith Scene Asset importiert [Epi22d].

   In diesem Testszenario wird die zweite Variante untersucht und eine uatasmith-Datei aus Rhino exportiert. Nur bei Frage 1.d) werden beide Varianten betrachtet, da es um Vor- und Nachteile von mehreren Möglichkeiten geht.

   b) **Welche 3D-Assets sollen ausgetauscht werden und wie werden sie in den Programmen repräsentiert?**

   Es werden Geometrien und Materialien ausgetauscht. Die Beispiel-Szene in Rhino ist ausschließlich aus NURBS und Kurven aufgebaut, die in Layer und Blöcke eingeteilt sind. Unreal Engine arbeitet mit triangulierten Objekten und Szenengraphen. Die Materials in der Rhinoszene haben die Eigenschaften aus Abbildung 8.2, wohingegen Unreal Engine-Materialien andere Bezeichnungen für die Parameter haben (siehe Abbildung 3.3).

      i. **Braucht man ein intermediäres Dateiformat?**

      Die Szene wird als udatasmith-Datei exportiert.

      ii. **Unterstützt eines der Programme das Format des anderen?**

      Das trifft für diese Variante von Datasmith nicht zu.

   c) **Soll bzw. kann der Austausch unidirektional oder bidirektional stattfinden?**

   Der Austausch findet unidirektional statt.

   d) **Zu welchem Zeitpunkt finden Asset-Konvertierungen statt? Gibt es Vorteile oder Nachteile bei mehreren möglichen Zeitpunkten?**

   In dem Testszenario mit dem Datasmithimporter werden die 3D-Objekte beim Szenenimport in die Unreal Engine bei Bedarf tesseliert. Es kann entschieden werden, ob NURBS trianguliert werden sollen, oder ob die Geometrien bereits trianguliert sind und man Rhino-Meshes importiert. Für die Triangulierung von CAD-Daten gibt es einige Einstellungen, z.B. wie stark die Normalen im Winkel voneinander abweichen, die maximale Kantenlänge von Polygonen und eine Stitching Technik. Andere Daten, wie die Szenenhierarchie und Materialien werden auch beim Import konvertiert.

   Bei der Variante, die Szene aus Rhino zu exportieren, werden die Geometrien beim Export mit den Rhino-Tesselierungseinstellungen konvertiert.

Bei einem direkten Versuch mit den Standardeinstellungen von jeweils dem Rhino-Datasmithexporter und dem Unreal-Datasmithimporter, haben die Meshes bei der Exportvariante deutlich weniger Polygone, ohne einen sichtbaren Detailverlust. Der Rhinoexporter erstellt insgesamt ca. 500000 Polygone. Mit dem Unreal Engine-Importer ergeben die Meshfaces in der Summe ca. 1200000 Polygone. Die Materialien, die Anzahl an Meshes und der Szenengraph sind bei beiden Versuchen identisch.

Ein Vorteil des Szenarios mit dem Datasmith-Import von nativen CAD-Dateien ist eine mögliche Retesselierung einzelner Meshes mit individuellen Konfigurationen [Epi22f].

e) **Ist der Zugriff von beiden Programmen auf ein Asset synchron?**

Der Zugriff auf die Assets ist asynchron.

Mit dem Datasmithexporter-Workflow gibt es aber die Möglichkeit, eine DirectLink-Connection zwischen Programmen lokal aufzubauen. Das wurde im Rahmen dieser Arbeit nicht getestet [Epi22g].

2. **Exporter**

a) **Wie viel Kontrolle hat man über die gespeicherten Daten? Welche Einstellungen stehen zu Verfügung?**

Der Datasmithexporter in Rhino verwendet benutzerdefinierten Tesselierungseinstellungen in Rhino, um die CAD-Modelle zu konvertieren. Es gibt kein Datasmith-Panel, mit dem man festlegt, welche Assets übernommen werden. Aber Rhino selbst unterstützt beim Exportieren die Auswahl, nur Geometrien auszuspeichern, Texturen mitzunehmen und Plugin-Daten zu speichern.

b) **Wie lange dauert der Export?**

Der Export dauert eine Minute.

c) **Welche Daten oder Datenstrukturen werden nicht exportiert, obwohl das Dateiformat sie beinhalten könnte?**

Rhino-Gruppen werden nicht übersetzt [Epi22h].

3. **Importer**

a) **Wie ist der Import von Assets gestaltet?**

Der Import einer udatasmith-Datei bietet die Auswahl, welche Assets importiert werden. Es können Geometrien, Materialien, Lichtquellen, Kameras und Animationen ausgewählt werden. Außerdem gibt es Einstellungen für die Erstellung von Lightmaps. Diese Einstellungen werden gespeichert und bei einem Reimport werden sie nochmals angewendet.

i. **Wie destruktiv ist ein Reimport von aktualisierten Daten? Erkennt der Importer, z.B. anhand des Namens des Assets oder anderer Metadaten, dass ein bereits bestehendes Objekt durch eine neue Version ersetzt werden soll und erhält Eigenschaften, die in dem Importprogramm hinzugefügt wurden?**

Datasmith hat eine sehr detaillierte Dokumentation zum Reimportprozess. Änderungen an Assets einer Datasmithszene durch Unreal Engine werden getrackt. Bei einem Reimport der gleichen Szene oder eines Assets, werden nicht veränderte Assets in Unreal Engine mit den Änderung aus der udatasmith-Datei aktualisiert. Einige getrackte Assetänderungen, z.B. Transformationen oder ein neues Material, werden beibehalten.

ii. **Gibt es die Möglichkeit, Schritte automatisch auszuführen um das Asset zu integrieren, z.B. für die Zuordnung von Programmskripten, die Substitution von Assets oder das Zusammenführen von Assets?**

In Unreal Engine selbst gibt es das Visual Dataprep-Plugin, das in Verbindung mit dem Datasmithimporter verwendet werden kann, um die in der Frage genannten Schritte zu automatisieren.

b) **Wie lange dauert der Import?**

Der initielle Import der Daten dauert ca. 2 Minuten.

c) **Welche Daten oder Datenstrukturen werden nicht importiert, obwohl sie von dem Programm interpretiert werden könnten?**

Das ist nicht bekannt.

4. **Dateiformat**

a) **Welche wichtigen Daten und Datenstrukturen können mit dem Dateiformat nicht gespeichert werden, z.B. keine Hierarchie, keine Metadaten, keine Animationen?**

Die Kurven werden nicht aus der Rhinoszene übernommen.

b) **Enthält es redundante Daten?**

Das ist nicht bekannt.

c) **Wie viel Speicherplatz nimmt die Datei ein?**

Die Datei ist 3,2 MB groß, der dazugehörige Assetordner 68,8 MB.

d) **Wird das Dateiformat während der Laufzeit der Anwendung ausgelesen? Wenn ja, wie performant ist das Auslesen der Daten?**

Man kann Datasmith-Dateien in ein Projekt integrieren, sodass eine udatasmith-Datei während der Laufzeit gelesen wird [Epi22i]. Das Feature wurde nicht getestet. Ansonsten wird das übliche Datasmith Scene Asset verwendet.

5. **Assets**

a) **Szenenhierarchie und Metadaten**

i. **Wie konsistent ist die Benennung von Assets?**

Die Benennung der 3D-Objekte von Rhino wird übernommen, allerdings können Umlaute nicht korrekt dargestellt werden in Unreal Engine, werden jedoch richtig in udatasmith gespeichert. Die automatische Benennung ist entweder „ON_Brep“ „ON_Extrusion“ plus eine inkrementelle Zahl. Layernamen werden übernommen und Blöcke haben auch ihren ursprünglichen Namen plus eine inkrementelle Zahl. Die Assets können gleiche Namen im Content Browser von Unreal Engine haben.

ii. **Wie wird die Szene hierarchisiert und wie werden verschiedene Hierarchiemodelle übersetzt?**

Layer und Blöcke werden in Unreal Engines Szenengraph als Actors übersetzt, die als Elternknoten der Meshes dienen. Jeder instanziierte Block ist ein individueller Actor.

b) **Koordinatensystem**

i. **Wird die Up-Axis korrekt übernommen?**

Ja, die Up-Axis ist allerdings die gleiche in Rhino und in Unreal Engine. Es ist unbekannt, ob die Szene bei unterschiedlichen Up-Axes rotiert würde.

ii. **Wird die Einheit korrekt übernommen?**

Ja, die Einheit wird korrekt übernommen.

iii. **Wie werden Transformationen umgesetzt?**

Die Transformationen liegen direkt auf den StaticMeshActors in Unreal Engine, alle Actors haben ihre Position im Ursprung. Die Transformation der Meshes wird vom Rhino-Ursprung aus berechnet.

c) **Oberflächen**

i. **Werden Vertex-Attribute wie Normalen, Texturkoordinaten, Vertex-Farben etc. korrekt übernommen?**

Die Normalen und Texturkoordinaten werden richtig übernommen.

ii. **Ist eine Tesselierung der Oberflächen notwendig?**

Ja, eine Tesselierung ist notwendig.

A. **Kann man Teile des Objektes mit unterschiedlichen Einstellungen tesselieren?**

Nein, die Objekte werden beim Export global mit den Rhino-Einstellungen tesseliert.

B. **Welche Einstellungen werden zum Tesselieren bereitgestellt?**

Der Rhinotesselierer hat zwei Stanardeinstellungen und die Möglichkeit zu benutzerdefinierten Tesseliereinstellungen. In Abbildung 8.3 werden die verfügbaren Einstellungen abgebildet.

C. **Müssen die Polygone reduziert werden?**

Ja, die tesselierten Objekte haben insgesamt über 500000 Polygone mit der Standardeinstellung „Gezackt und schneller“ des Rhinotesselierers. Die hohe Anzahl an Polygonen kommt vor allem durch die dichte Tesselierung von organischen Flächen, wie z.B. kurvige Rohre. Flache Oberflächen werden nicht unnötigerweise tesseliert. Mit den anderen Einstellungsmöglichkeiten konnten in den durchgeführten Versuchen keine Einstellungen gefunden werden, die weniger Polygone haben und gleichzeitig zufriedenstellende Ergebnisse über das ganze Mesh hinweg erzielen.

D. **Müssen Normalen invertiert werden?**

Nein, die Normalen zeigen in die richtige Richtung.

E. **Entstehen Deformationen wie Überlappungen oder Spalten zwischen Patches?**

Über die Oberflächen des Rhino-Modells hinweg wurden keine Überlappungen festgestellt. Spalten gibt es zwischen verschiedenen Objekten, die im Rhino-Modell nicht zusammenliegen.

F. **Können Vertices zusammengeführt werden? Wenn ja, wie differenziert geschieht das zwischen unterschiedlich detaillierten Objekten?**

Nein, es gibt keine Option dafür in dem Austausch- und Konvertierungsprozess.

d) **Materials**

i. **Auf welche Weise werden Materials übernommen?**

Die Materials werden als Materialinstanzen in Unreal Engine importiert, wobei die einzelnen Materials von automatisch zusammengefassten Parent Materials erben. Alle Materials sind nur einseitig. Da das beim Rendern dieser Szene durch Backface-Culling zu durchsichtigen Rohrinnenseiten führt, die eigentlich sichtbar sein sollten, müssen die Parent-Materialien manuell als zweiseitig definiert werden.

ii. **Welche Oberflächenwerte bzw. Shaderoperationen können übersetzt werden?**

Die Rhino-Farbe wird als Basecolor gesetzt. Es gibt in der Szene eine Transparenz-Maske, die in Unreal-Engine als Opacity gesetzt wird. Das Reflexionsvermögen wird dem Metallic-Parameter übergeben und Glänzendes Finish wird den Beobachtungen zufolge als invertierter Wert (1 - Glänzendes Finish) der Roughness zugewiesen.

Das UV-Tiling der Textur ist nicht ganz korrekt. Die Textur wirkt daher in eine Richtung gestaucht. Insgesamt ist das Ergebnis in Ordnung, die Materialien sehen alle rauer aus als im Rhino-Renderer und die Opacity wird nicht nur maskiert, sondern das ganze Material ist ein wenig transparent.

iii. **Wie volatil ist die Material- bzw. Texturreferenzierung?**

Die Material- und Texturreferenzierung hat keine Fehler aufgewiesen. Der Datasmith-Assetordner sollte allerdings im gleichen Ordner wie die udatasmith-Datei bleiben.

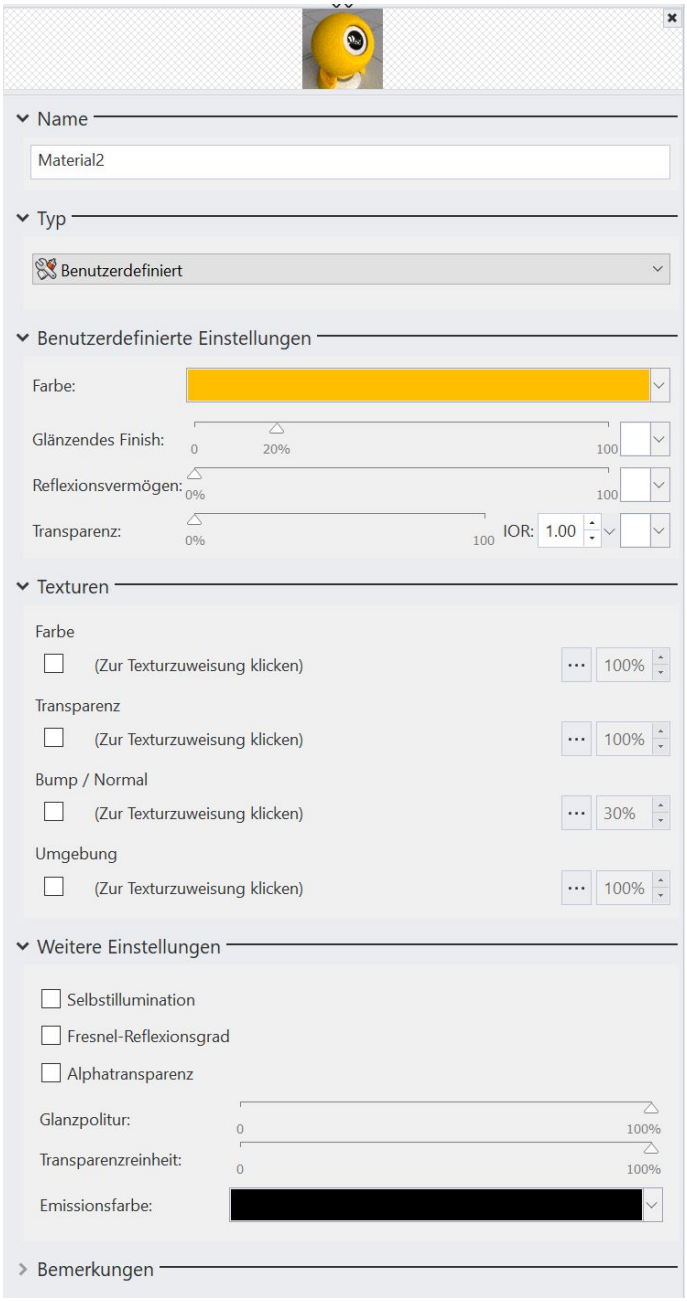

Abbildung 8.2: Rhino-Material aus der Demonstratorszene

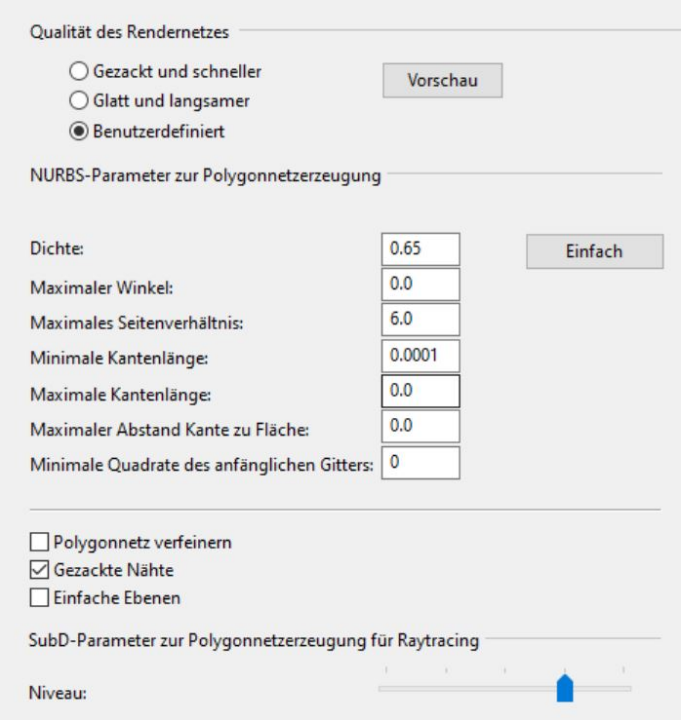

Abbildung 8.3: Rhino-Tesseliereinstellungen

8.2 Rhino zu Unreal Engine Export mit Omniverse und USD

1. **Austausch**

   a) **Gibt es zwsichen den Programmen eine passende Lösung, um die zu übertragenden 3D-Assets in ein Format zu exportieren, das von dem nächsten Importierer gelesen werden kann?**

   Über den Omniverse Connector für Rhino wird der Export auf einen Nucleus-Server vorgenommen. Die Szene wird also in eine USD-Stage geschrieben, die durch den Omniverse Unreal Engine Connector ausgelesen wird.

   b) **Welche 3D-Assets sollen ausgetauscht werden und wie werden sie in den Programmen repräsentiert?**

   Siehe erstes Beispiel.

      i. **Braucht man ein intermediäres Dateiformat?**

      Ja, es wird die von Omniverse erweiterte Version von USD verwendet.

      ii. **Unterstützt eines der Programme das Format des anderen?**

      Nein, die beiden Programme sind sehr unterschiedlich in der Beschreibung ihrer Assets.

c) **Soll bzw. kann der Austausch unidirektional oder bidirektional stattfinden?**

Trotz der Bidirektionalität des Unreal Engine Connectors kann der Austausch der Daten ausschließlich unidirektional von Rhino zu Unreal Engine erfolgen, da der Rhino Connector nur als Publisher-Client agieren kann. Er kann keine Änderungen von dem Nucleus Server lesen [NVI22c].

d) **Zu welchem Zeitpunkt finden Asset-Konvertierungen statt? Gibt es Vorteile oder Nachteile bei mehreren möglichen Zeitpunkten?**

Bei einem Export durch den Rhino Connector werden die CAD-Daten tesseliert und als Meshes gespeichert. Die Rhino Materials werden beim Exportieren in ein OmniPBR-MDL konvertiert.

e) **Ist der Zugriff von beiden Programmen auf ein Asset synchron?**

Mehrere Clients können auf ein Asset auf dem Nucleus Server synchron zugreifen. Da der Rhino Connector nur unidirektional ist, empfiehlt sich allerdings kein direkter Zugriff auf die exportierte USD-Szene, denn das führt sonst zu einem destruktiven Workflow. Der korrekte Workflow wäre laut einer Präsentation von NVIDIA zu dem Rhino Connector, eine zweite USD-Szene zu erstellen und die exportierte Szene lediglich zu referenzieren. Dadurch können Werte überschrieben werden, ohne dass die ursprüngliche Datei verändert wird [Rob21]. Der Zugriff auf eine Datei auf dem Omniverse Server durch Unreal Engine kann generell live oder asynchron sein.

2. **Exporter**

a) **Wie viel Kontrolle hat man über die gespeicherten Daten? Welche Einstellungen stehen zu Verfügung?**

Der Rhino-Connector tesseliert die Objekte beim Export automatisch. Für den Export von Geometrien gibt es die Möglichkeiten, entweder jedes Objekt einzeln abzuspeichern oder Objekte zusammenzufassen. Der Unterschied der beiden Möglichkeiten in der Anzahl an Meshes ist 3699 zu 809 Meshes. Allerdings kann man zusammengefasste Objekte später nicht mehr einzeln auswählen.

Für Materials gibt es einige Konfigurationsmöglichkeiten. Man hat die Möglichkeit ein Materialmapping vorzunehmen, benutzerdefinierte Rhino Materials zu übernehmen, Materialien pro Layer zu übertragen und den Layernamen als Material zu verwenden. Mit unterschiedlichen Einstellungen werden unterschiedliche Ergebnisse erzielt. Ohne jegliche aktivierte Einstellung, existiert pro Mesh ein Material in der USD-Szene. Wenn man Rhino-Materials übernimmt, reduziert sich die Anzahl der Materialien auf die tatsächliche Anzahl der Rhino-Materialien. Dabei ist es wichtig, dass jedes Rhino-Material einen individuellen Namen hat, sonst werden sie nicht übernommen.

b) **Wie lange dauert der Export?**

Der Export dauert ca. fünf Minuten.

c) **Welche Daten oder Datenstrukturen werden nicht exportiert, obwohl das Dateiformat sie beinhalten könnte?**

Die Kurven in der Szene werden nicht exportiert und es werden auch keine Texturen übertragen.

3. **Importer**

a) **Wie ist der Import von Assets gestaltet?**

Da man auf einer Datei auf einem Server arbeitet und es möglich ist, eine Live-Sync-Session mit dem Nucleusserver zu starten, sind Iterationen eines Datei-Imports bei Änderungen nicht notwendig. Wenn man neue Änderungen ohne eine Live-Synchronisierung importieren möchte, öffnet man die Szene nochmals und ist auf dem neuesten Stand, überschreibt damit aber seine lokalen Änderungen. Wenn man seine lokalen Änderungen speichert, überschreibt man die ungespeicherten Änderungen anderer Clients [NVI22d].

i. **Wie destruktiv ist ein Reimport von aktualisierten Daten? Erkennt der Importer, z.B. anhand des Namens des Assets oder anderer Metadaten, dass ein bereits bestehendes Objekt durch eine neue Version ersetzt werden soll und erhält Eigenschaften, die in dem Importprogramm hinzugefügt wurden?**

Veränderte lokale Eigenschaften werden bei einem Reload überschrieben, wenn man nicht mit Live-Sync arbeitet.

ii. **Gibt es die Möglichkeit, Schritte automatisch auszuführen um Assets zu integrieren, z.B. für die Zuordnung von Programmskripten, die Substitution von Assets oder das Zusammenführen von Assets?**

Es gibt die Möglichkeit Material-Mapping und Objektsubstitutionen anhand von benutzerdefinierten CSV-Dateien zu machen, das passiert allerdings beim Export aus Rhino [Rob21]. Auch das Zusammenführen von Objekten kann man beim Rhino-Export festlegen. Auf welcher Grundlage das geschieht, ist jedoch nicht einsichtig.

b) **Wie lange dauert der Import?**

In wenigen Sekunden ist die Szene geladen.

c) **Welche Daten oder Datenstrukturen werden nicht importiert, obwohl sie von dem Programm interpretiert werden könnten?**

Alle Daten aus der USD-Szene auf dem Nucleusserver wurden in Unreal Engine übernommen.

4. **Dateiformat**

a) **Welche wichtigen Daten und Datenstrukturen können mit dem Dateiformat nicht gespeichert werden, z.B. keine Hierarchie, keine Metadaten, keine Animationen?**

Auf USD bezogen könnten theoretisch alle Assets, die in der Rhino-Szene vorkommen, verlustlos gespeichert werden. USD unterstützt NURBS, Kurven, Materialien und Texturen und könnte mit Metadaten erweitert werden.

b) **Enthält es redundante Daten?**

Ja, Meshes bekommen eine Display-Color als Attribut zugewiesen, die in Unreal nicht zum Rendern verwendet wird. Zusätzlich wird in den zugewiesenen Materialien die Farbe eines Meshes als Albedo-Farbe gespeichert.

c) **Wie viel Speicherplatz nimmt die Datei ein?**

Die USDA-Datei ist ca. 100 MB groß.

d) **Wird das Dateiformat während der Laufzeit der Anwendung ausgelesen? Wenn ja, wie performant ist das Auslesen der Daten?**

Die USD-Daten des Nucleusserver werden in lokale intermediäre Unreal-Assets überführt und gecached. Laut der Dokumentation des Omniverse Unreal Enigne Connectors werden USD-Dateien nicht in das native Unreal Engine-Format konvertiert, MDL-Dateien werden aber in Unreal Engine-Materials konvertiert [NVI22d].

5. **Assets**

a) **Szenenhierarchie und Metadaten**

i. **Wie konsistent ist die Benennung von Assets?**

3D-Geometrien können in Rhino benannt werden, allerdings sind die Objekte in dieser 3dm-Datei unbenannt. Einige Rhino-Materials in der Szene besitzen einen Namen, andere sind unbenannt.

In der Unreal Engine haben die 3D-Objekte, die über den Unreal Engine Connector importiert werden einen Namen, der aus „Mesh_“ und einem Hashwert besteht oder aus Mesh und einer inkrementellen Zahl.

Die Speicherung von Umlauten in Namen ist nicht möglich.

ii. **Wie wird die Szene hierarchisiert und wie werden verschiedene Hierarchiemodelle übersetzt?**

Bei der Exportauswahl, einzelne Objekte zu exportieren, werden die Layer aus Rhino in den World Outliner in Unreal Engine übernommen. Blöcke sind den Layern nicht untergeordnet, sondern in einer nebengeordneten Hierarchie aufgeführt.

Beim Exportieren ohne diese Auswahl werden Meshes innerhalb von Blöcken zusammengeführt. Dabei wird nur die Layerstruktur übernommen, denen alle Meshes, auch die aus Blöcken, untergeordnet sind. Allerdings werden auch andere Meshes außerhalb von Blöcken in der Layerstruktur zusammengeführt. Auf welcher Basis das passiert, ist nicht einsichtig.

b) **Koordinatensystem**

i. **Wird die Up-Axis korrekt übernommen?**

Die Up-Axis in Rhino ist die Z-Achse, das wird in der USD-Datei gespeichert. Auch Unreal Engine hat ein Z-Up-Koordinatensystem. Das wird korrekt übernommen und in der USD-Datei gespeichert

ii. **Wird die Einheit korrekt übernommen?**

Die Einheit wird korrekt übernommen. Der Exporter speichert die Einheit der Rhino-Szene allerdings nicht in der USD-Datei.

iii. **Wie werden Transformationen umgesetzt?**

Transformationen liegen direkt auf den Meshes. Die Transformation wird vom Rhino-Ursprung ausgehend berechnet.

c) **Oberflächen**

i. **Werden Vertex-Attribute wie Normalen, Texturkoordinaten, Vertex-Farben etc. korrekt übernommen?**

Normalen werden korrekt übernommen. Andere Attribute wurden nicht getestet.

ii. **Ist eine Tesselierung der Oberflächen notwendig?**

A. **Kann man Teile des Objektes mit unterschiedlichen Einstellungen tesselieren?**

Nein, die Objekte werden beim Export global mit den Rhino-Einstellungen tesseliert.

B. **Wie detailliert sind die Konfigurationsmöglichkeiten zum Tesselieren?**

Siehe erstes Beispiel.

C. **Müssen die Polygone reduziert werden?**

Siehe erstes Beispiel. Wobei die tesselierten Objekte mit der Standardeinstellung „Gezackt und schneller“ des Rhinotesselierers hier insgesamt über 1200000 Polygone haben.

D. **Müssen Normalen invertiert werden?**

Nein, die Normalen zeigen in die richtige Richtung.

E. **Entstehen Deformationen wie Überlappungen oder Spalten zwischen Patches?**

Über die Oberflächen des Rhino-Modells hinweg wurden keine Überlappungen festgestellt. Spalten gibt es nur zwischen Objekten, die im Rhino-Modell nicht zusammenliegen.

F. **Können Vertices zusammengeführt werden? Wenn ja, wie differenziert geschieht das zwischen unterschiedlich detaillierten Objekten?**

Nein, es gibt keine Option dafür in dem Austausch- und Konvertierungsprozess.

d) **Materials**

i. **Auf welche Weise werden Materials übernommen?**

   Mit dem Omniverse Connector werden die Rhino-Materialien in Unreal Engine Materialinstanzen eines OmniPBR-Materials konvertiert. Somit werden die Materialien in Unreal Engine mit den Omniverse MDL-Materialparametern erweitert. Diese OmniPBR-Materialien haben weitaus mehr Parameter, als ein Rhino-Material. Eine Auflistung aller OmniPBR-Materialeigenschaften stellt die Dokumentation der Create App bereit (siehe [NVI22e]).

ii. **Welche Oberflächenwerte bzw. Shaderoperationen können übersetzt werden?**

   Die Farbe wird als Albedoparameter übersetzt, das Reflexionsvermögen der Rhino-Materials wird dem Metallic-Parameter übergeben und glänzendes Finish aus der Rhino-Szene wird als Specular-Wert gespeichert.

iii. **Wie volatil ist die Material- bzw. Texturreferenzierung?**

   Die Texturen werden, wie bereits erwähnt, nicht übernommen. Materialien werden zuverlässig in einem MDL-Ordner referenziert.

8.3 Schlussfolgerung

Die Austauschszenarien haben Ähnlichkeiten in verschiedenen Punkten. Sie benutzen beide ein intermediäres Dateiformat, das durch ein Plugin in Rhino exportiert wird. Für beide Exporte werden die Rhino-Einstellungen zur Konvertierung der CAD-Modelle in Meshes verwendet. Die Übernahme der Orientierung, Einheit und Transformationen führt mit beiden Varianten zu den gleichen Ergebnissen.

Der größte Unterschied liegt in der Art und Weise, wie der Zugriff auf die Dateien geschieht. Dadurch, dass alle Clients in der Omniverse-Variante an einer Datei arbeiten, ist der Aktualisierungsvorgang sehr unterschiedlich zu Datasmith. Vorteile von Datasmith im Vergleich zum Omniverse Connector sind der kontrollierte Reimport und konstruktive Konfliktansätze. Für den Omniverse Connector in Unreal Engine sollte klar definiert werden, wie mit veränderten Objekten im Falle eines Konflikts umgegangen wird. Die Überschreibung der Workfile bei einer Speicherung macht die Pipeline destruktiv, wenn andere Clients gerade daran arbeiten, und eine sehr enge Kommunikation der Entwickler:innen bei einer Aktualisierung ist erforderlich. Der Omniverse Connector ist bei einem Reload der USD-Stage nicht änderungserhaltend. Die Integrierung der Assets in den Unreal Engine Editor ist generell mit Datasmith besser, da andere Plugins, z.B. Visual Dataprep, in Verbindung verwendet werden können.

Die Speicherzeit der Omniverse USD-Dateien waren in diesem Testsetup ein wenig länger. Dafür wurde die USD-Stage schneller ausgelesen von Unreal Engine.

Mit den beiden Austauschmöglichkeiten wird die Szenenhierarchie der Rhinoszene mit den Layerebenen und Blöcken mithilfe von Actors leicht unterschiedlich nachgebaut. Während Datasmith Blöcke unter den Layerebenen einordnet, werden Blöcke mit Omniverse entweder der Layer-Struktur nebengeordnet oder bei der Zusammenführung von Meshes werden die Blöcke als ein Mesh den Layern untergeordnet.

Obwohl in beiden Szenarien die Tesselierung der Objekte mit der gleichen Rhino-Einstellung „Gezackt und schneller“ durchgeführt werden, hat die Unreal Engine-Szene mit den Omniverseassets mehr als doppelt so viele Polygone wie die Datasmith-Szene. In beiden Varianten werden organische Formen jedoch mit den Standardeinstellungen zu detailliert tesseliert. Mit benutzerdefinierten Einstellungen der Rhinopolygonnetzerzeugung wurden schlechtere Ergebnisse sowohl in der Anzahl an Polygonen als auch in der visuellen Qualität verzeichnet.

Mit der Zusammenführung der Assets beim Omniverse-Export wird die Anzahl an Meshes auf ein Viertel reduziert. Es ist allerdings nicht einsichtig, auf welcher Grundlage die Assets zusammengeführt werden.

Datasmith kann im Gegensatz zum Omniverse Connector die Texturen übertragen. Andererseits ist ein OmniPBR-Material von Omniverse standardmäßig zweiseitig, was in dieser Szene von Bedeutung ist. Für beide Wege ist eine Nachbearbeitung der Materialien notwendig.

8.4 Limitierungen des Kriterienkatalogs

Die Abarbeitung des Kriterienkatalogs deckt bereits wichtige Punkte im Bezug auf die Workflows und die Übernahme von Assets mit den beiden Austauschszenarien ab. Er beleuchtet aber noch nicht alle Seiten einer Produktionspipeline. Ein wichtiger Aspekt ist beispielsweise noch die Versionierung von Assets. Das wurde vor allem bei dem Test mit dem Unreal Engine Omniverse Connector sichtbar. Es gibt zwar die sogenannten Checkpoints, aber eine Konfliktlösung von Änderungen an einer Workfile wird nicht geboten.

Ein weiterer Aspekt, der in dem Kriterienkatalog noch nicht behandelt wird, sind Physics bzw. physikalische Parameter eines Objektes. Der Omniverse Connector in Unreal Engine fügt den Meshes beispielsweise standardmäßig eine Collision hinzu, wohingegen Datasmith den Objekten keine Collision zuweist.

9 Fazit

9.1 Anwendbarkeit des Kriterienkatalogs

In dem Kriterienkatalog sind die Fragen teilweise sehr weitläufig gestellt und die Initiative, die Fragen in Tiefe zu beantworten, hängt von den Bearbeiter:innen ab. Bei einer oberflächlichen Bearbeitung lassen sich vielleicht weniger Punkte miteinander vergleichen, da sie an Detail und Aussagekraft verlieren.

Dadurch, dass keine bestimmten Metriken gestellt werden, mit denen die Fragen beantwortet und evaluiert werden können, ist die Bewertung eines Austausches auf der Grundlage von vordefinierten Attributen oder statistischen Werten nicht möglich. Mit dem Kriterienkatalog ist es vielmehr möglich, ein persönliches Fazit zu entnehmen und zu kontrollieren, ob die projektspezifischen Anforderungen an einen Austausch mit den untersuchten Mitteln umsetzbar sind.

Der Fragenkatalog wurde nur mit dem Beispiel einer Architekturvisualisierung ohne Animationen getestet. Für eine gänzliche Bewertung der Qualität des Kriterienkatalogs sollten noch weitere Testszenarien mit Animationsassets und anderen Anwendungsfällen durchgeführt werden.

Die Dokumentation der Plugins und die Einsichtigkeit von USD-Daten und udatasmith-Dateien waren sehr hilfreich bei der Beantwortung der Fragen des Kriterienkatalogs. Ein Tool, das die Rohdaten einer Datei leserlich ausgibt, kann generell bei der Evaluierung und Problemfindung hilfreich sein.

9.2 Kommentar zur NVIDIA Omniverse Plattform

Die Cloud-Lösung zur Speicherung der Asstes und die synchrone Kollaboration an den Szenen ist grundsätzlich ein spannender Ansatz. Die Live-Synchronisierung zwischen Unreal Engine und den Omniverse Renderern funktioniert sehr schnell und das Checkpoint-Konzept sichert die Historie einer Datei, sodass man ältere Versionen wiederherstellen kann.

In einer realen Entwicklung mit mehreren Clients bzw. Entwickler:innen, die an einer Szene arbeiten, könnten mit dem jetzigen Stand der Kollaborationsplattform aber Probleme entstehen, sobald mehrere Clients nicht synchronisiert an einer Datei arbeiten.

Einige Konfigurationen, die verschiedene Operationen beim Import oder Export durchführen, sind nicht deutlich definiert. Dass der Omniverse Rhino Connector beispielsweise die Rhinotesselier-einstellungen verwendet, wurde durch Testen von exportierten Szenen entdeckt. Manche andere Fragen des Kriterienkatalogs konnten nicht im Detail erklärt werden, z.B. auf welcher Grundlage manche Objekte zusammengeführt werden.

Mit dem Rechnersetup dieser Arbeit stürzten außerdem Teile der Omniverseplattform einige Male ab. Sowohl die Omniverse Create App als auch die Kombination aus 3D-Applikation und Connector waren nicht immer stabil.

9.3 Ausblick

Die Anwendungs- und Pluginlandschaft von DCC-Applikationen und Echtzeitrendering Engines ist sehr groß und heterogen. Es werden zahlreiche Lösungen zu unterschiedlichen Aspekten des 3D-Asset-Austausches angeboten, die auf verschiedene Anwendungsfälle passen. Viele Firmen in der interaktiven Medienbranche und die Filmbranche benutzen unter anderem eigene 3D-Programme und entwickeln eigene Lösungen, die nur teilweise veröffentlicht werden. Mehr Daten zum Arbeitsprozess der Entwickler:innen und zu den Anforderungen an 3D-Programmen sowie Import und Exportszenarien könnten zukünftig dabei helfen, Probleme der 3D-Assetpipelines zu abstrahieren und besser zu kategorisieren.

Cloud-Lösungen für die Zusammenarbeit an Projekten zwischen unterschiedlichen Programmen ist eine mögliche Entwicklung für Echtzeitrendering-Projekte der Zukunft. Echtzeitupdates in der Produktion ist eine naheliegende Weiterentwicklung für diese Branche. Der Idealfall dafür wäre ein einheitliches Verständnis von den ausgetauschten Daten zwischen Programmen. Eine wünschenswerte Funktion für die Arbeit an einer Datei ist die Erhaltung von Datenänderungen aller Mitwirkenden, sobald ein Konflikt aufkommt, und wohldefinierte Lösungen für mögliche Konflikte.

Literaturverzeichnis

[Ado19] Adobe. *Bakers Settings*. 2019. URL: https://substance3d.adobe.com/documentation/bake/bakers-settings-172818452.html (zitiert auf S. 21).

[Ado21] Adobe. *Bake Mesh Maps*. 2021. URL: https://helpx.adobe.com/substance-3d-painter/using/baking.html (zitiert auf S. 21, 22).

[Aut] Autodesk FBX. *FBX SDK Programmer's Guide*. URL: http://docs.autodesk.com/FBX/2014/ENU/FBX-SDK-Documentation/ (zitiert auf S. 25).

[Aut18] Autodesk Inc. *Alembic (ABC) Files*. Dez. 2018. URL: https://knowledge.autodesk.com/support/3ds-max/learn-explore/caas/CloudHelp/cloudhelp/2019/ENU/3DSMax-Data-Exchange/files/GUID-D80A02B6-BC7B-4070-A959-94EC5FCA22F8-htm.html (zitiert auf S. 26).

[Aut21] Autodesk. *Arnold for Maya User Guide. Lights*. 2021. URL: https://docs.arnoldrenderer.com/display/A5AFMUG/Lights (zitiert auf S. 20).

[Aut22] Autodesk Inc. *Look development using the Hypershade*. 2022. URL: https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2020/ENU/Maya-LightingShading/files/GUID-447C9ADA-A285-483E-A93D-6ED1B2792D5F-htm.html (zitiert auf S. 20).

[Bab] Babylon.js. *Introduction to Physically Based Rendering*. URL: https://doc.babylonjs.com/divingDeeper/materials/using/introToPBR (zitiert auf S. 19).

[BFS08] M. Barnes, E. L. Finch, Sony Computer Entertainment Inc. *COLLADA – Digital Asset Schema Release 1.5.0. Specification*. 2008. URL: https://www.khronos.org/files/collada_spec_1_5.pdf (zitiert auf S. 27).

[Blea] Blender. *Blender 2.80 Manual. glTF 2.0*. URL: https://docs.blender.org/manual/en/2.80/addons/io_scene_gltf2.html (zitiert auf S. 15, 26).

[Bleb] Blender. *Blender 2.80 Manual. Wavefront OBJ*. URL: https://docs.blender.org/manual/en/2.80/addons/io_scene_obj.html (zitiert auf S. 25).

[Blec] Blender Foundation. *UV Unwrapping*. URL: https://www.blender.org/features/modeling/ (zitiert auf S. 19).

[Ble10] Blender. *FBX binary file format specification*. Aug. 2010. URL: https://code.blender.org/2013/08/fbx-binary-file-format-specification/ (zitiert auf S. 25).

[Ble22a] Blender. *Blender 3.0 Manual. Mesh - Structure - Faces*. 2022. URL: https://docs.blender.org/manual/en/latest/modeling/meshes/structure.html (zitiert auf S. 18).

[Ble22b] Blender. *Blender 3.0 Manual. Shader Editor*. 2022. URL: https://docs.blender.org/manual/en/latest/editors/shader_editor.html (zitiert auf S. 20).

[Ble22c] Blender. *Blender 3.0 Manual. Animation & Rigging*. 2022. URL: https://docs.blender.org/manual/en/latest/animation/introduction.html (zitiert auf S. 21).

[Ble22d] Blender. *Blender 3.0 Manual. Bone Constraints*. 2022. URL: https://docs.blender.org/manual/en/2.80/animation/armatures/posing/bone_constraints/introduction.html (zitiert auf S. 22).

[Ble22e] Blender. *Blender 3.0 Manual. Animation - Constraints - Introduction*. 2022. URL: https://docs.blender.org/manual/en/latest/animation/constraints/introduction.html (zitiert auf S. 23).

[Ble22f] Blender. *Blender 3.0 Manual. Animation & Rigging - Shape Keys - Introduction*. 2022. URL: https://docs.blender.org/manual/en/latest/animation/shape_keys/introduction.html (zitiert auf S. 23).

[Dan17] S. v. d. S. Dan Sumaili. *Creating a Tools Pipeline for 'Horizon: Zero Dawn'*. 2017. URL: https://www.gdcvault.com/play/1024124/Creating-a-Tools-Pipeline-for (zitiert auf S. 15).

[Epi22a] Epic Games, Inc. *Unreal Engine 4.27 Documentation. Light Mobility*. 2022. URL: https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/LightingAndShadows/LightMobility (zitiert auf S. 20).

[Epi22b] Epic Games, Inc. *Unreal Engine 4.27 Documentation. Using Datasmith with CAD File Formats*. 2022. URL: https://docs.unrealengine.com/4.27/en-US/WorkingWithContent/Importing/Datasmith/SoftwareInteropGuides/CAD/ (zitiert auf S. 15, 18).

[Epi22c] Epic Games, Inc. *Unreal Engine 4.27 Documentation. Material Expression Nodes and Networks*. 2022. URL: https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphics/Materials/IntroductionToMaterials/ (zitiert auf S. 20).

[Epi22d] Epic Games, Inc. *Unreal Engine 4.27 Documentation. Datasmith Overview*. 2022. URL: https://docs.unrealengine.com/4.27/en-US/WorkingWithContent/Importing/Datasmith/Overview/ (zitiert auf S. 27, 50).

[Epi22e] Epic Games, Inc. *Unreal Engine 4.27 Documentation. About the Datasmith Import Process*. 2022. URL: https://docs.unrealengine.com/4.27/en-US/WorkingWithContent/Importing/Datasmith/Overview/ImportProcess/ (zitiert auf S. 27).

[Epi22f] Epic Games, Inc. *Unreal Engine 4.27 Documentation. Retessellating CAD Geometry*. 2022. URL: https://docs.unrealengine.com/4.27/en-US/WorkingWithContent/Importing/Datasmith/SoftwareInteropGuides/CAD/Retessellation/ (zitiert auf S. 51).

[Epi22g] Epic Games, Inc. *Unreal Engine 4.27 Documentation. Using Datasmith Direct Link*. 2022. URL: https://docs.unrealengine.com/4.27/en-US/WorkingWithContent/Importing/Datasmith/Overview/UsingDatasmithDirectlink/ (zitiert auf S. 51).

[Epi22h] Epic Games, Inc. *Unreal Engine 4.27 Documentation. Using Datasmith with Rhino*. 2022. URL: https://docs.unrealengine.com/4.27/en-US/WorkingWithContent/Importing/Datasmith/SoftwareInteropGuides/Rhino/ (zitiert auf S. 51).

[Epi22i] Epic Games, Inc. *Unreal Engine 4.27 Documentation. Using Datasmith at Runtime*. 2022. URL: https://docs.unrealengine.com/4.27/en-US/WorkingWithContent/Importing/Datasmith/Overview/UsingDatasmithAtRuntime/ (zitiert auf S. 52).

[FDFH96] J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes. *Computer Graphics Principles and Practice*. 2nd in C. Addison Wesley Publishing Company, 1996. ISBN: 0-201-84840-6 (zitiert auf S. 17).

[ftr21] ftrack. *What is a Pipeline TD and how can I become one?* 2021. URL: https://www.ftrack.com/en/2021/09/what-is-a-pipeline-td.html (zitiert auf S. 11).

[GJJ16] D. van Gelder, P. Jeremias, D. Ju. *Real-Time Graphics in Film Production at Pixar*. 2016. URL: https://on-demand.gputechconf.com/siggraph/2016/video/sig1608-pol-jememias-dirk-van-gelder-david-yu-real-time-graphics-pixar.mp4 (zitiert auf S. 15).

[Hou19a] B. Houston. *glTF vs FBX: Which format should I use?* Nov. 2019. URL: https://www.threekit.com/blog/gltf-vs-fbx-which-format-should-i-use (zitiert auf S. 26).

[Hou19b] B. Houston. *OBJ 3D file format: When should you use it?* Nov. 2019. URL: https://www.threekit.com/blog/obj-3d-file-format-when-should-you-use-it (zitiert auf S. 25).

[Hou19c] B. Houston. *When should you use FBX 3D file format ?* Okt. 2019. URL: https://www.threekit.com/blog/when-should-you-use-fbx-3d-file-format (zitiert auf S. 25).

[IGDA21] International Game Developers Association. *Developer Satisfaction Survey 2021 - Summary Report*. 2021. URL: https://igda-website.s3.us-east-2.amazonaws.com/wp-content/uploads/2021/10/18113901/IGDA-DSS-2021_SummaryReport_2021.pdf (zitiert auf S. 11).

[Kar13] B. Karis. *Real Shading in Unreal Engine 4*. 2013. URL: https://de45xmedrsdbp.cloudfront.net/Resources/files/2013SiggraphPresentationsNotes-26915738.pdf (zitiert auf S. 20).

[Khr17] Khronos® Group Inc. *Portal:Post 2010*. 2017. URL: https://www.khronos.org/collada/wiki/Portal:Post_2010 (zitiert auf S. 27).

[Khr20] Khronos® Group Inc. *ColladaDOM 3. A cyberspace manifesto in brief*. 2020. URL: https://www.khronos.org/collada/wiki/ColladaDOM_3 (zitiert auf S. 27).

[Khr21] Khronos® 3D Formats Working Group. *glTF Specification*. Nov. 2021. URL: https://www.khronos.org/registry/glTF/specs/2.0/glTF-2.0.html (zitiert auf S. 26).

[Khr22] Khronos® Group Inc. *glTF RUNTIME 3D ASSET DELIVERY*. 2022. URL: https://www.khronos.org/gltf/ (zitiert auf S. 26).

[Lea20] J. Lear. *The Video Game Asset Pipeline - A Pattern Approach to Visualization*. Nov. 2020. URL: https://uwe-repository.worktribe.com/output/6844257 (zitiert auf S. 15).

[Lin17] J. Linietsky. *Why we should all support glTF 2.0 as THE standard asset exchange format for game engines*. Aug. 2017. URL: https://godotengine.org/article/we-should-all-use-gltf-20-export-3d-assets-game-engines (zitiert auf S. 26, 27).

[NVI22a] NVIDIA. *NVIDIA AT CES 2022*. Jan. 2022. URL: https://www.nvidia.com/en-us/events/ces/ (zitiert auf S. 13).

[NVI22b] NVIDIA. *Omniverse Connect Documentation. Version Control*. 2022. URL: https://docs.omniverse.nvidia.com/con_connect/con_connect/con_version-control.html (zitiert auf S. 29).

[NVI22c] NVIDIA. *Omniverse Connect Documentation. Connect Overview*. 2022. URL: https://docs.omniverse.nvidia.com/con_connect/con_connect/overview.html (zitiert auf S. 30, 57).

[NVI22d] NVIDIA. *Omniverse Connect Documentation. Unreal Engine 4*. 2022. URL: https://docs.omniverse.nvidia.com/con_connect/con_connect/ue4.html (zitiert auf S. 58, 59).

[NVI22e] NVIDIA. *Omniverse Create Documentation. Omniverse MDL Materials*. 2022. URL: https://docs.omniverse.nvidia.com/app_create/prod_materials-and-rendering/materials.html (zitiert auf S. 61).

[NVI22f] NVIDIA. *Omniverse Extensions Documentation. Extensions Overview*. 2022. URL: https://docs.omniverse.nvidia.com/prod_extensions/prod_extensions/overview.html (zitiert auf S. 30).

[NVI22g] NVIDIA. *Omniverse Nucleus Documentation. Introduction*. 2022. URL: https://docs.omniverse.nvidia.com/prod_nucleus/prod_nucleus/overview.html (zitiert auf S. 29).

[NVI22h] NVIDIA. *Omniverse Platform Documentation. Models, Materials and USD*. 2022. URL: https://docs.omniverse.nvidia.com/plat_omniverse/plat_omniverse/structure.html (zitiert auf S. 29).

[NVI22i] NVIDIA. *Omniverse Platform Documentation. Simulation*. 2022. URL: https://docs.omniverse.nvidia.com/plat_omniverse/plat_omniverse/simulation.html (zitiert auf S. 29).

[NVI22j] NVIDIA. *Omniverse Platform Documentation. Omniverse Structure*. 2022. URL: https://docs.omniverse.nvidia.com/plat_omniverse/plat_omniverse/structure.html (zitiert auf S. 30).

[NVI22k] NVIDIA Corporation. *NVIDIA Omniverse*. 2022. URL: https://www.nvidia.com/en-us/omniverse/ (zitiert auf S. 29).

[OCo17] K. O'Conor. *GPU Performance for Game Artists*. März 2017. URL: http://fragmentbuffer.com/gpu-performance-for-game-artists/ (zitiert auf S. 19).

[PAM10] P.Shirley, M. Ashikhmin, S. Marschner. *Fundamentals of Computer Graphics*. 3. Aufl. A K Peters/CRC Press, 2010. URL: https://doi.org/10.1201/9781439865521 (zitiert auf S. 21, 22).

[Pixa] Pixar Animation Studios. *USD API Reference. UsdGeom : USD Geometry Schema*. URL: https://graphics.pixar.com/usd/release/api/usd_geom_page_front.html (zitiert auf S. 32).

[Pixb] Pixar Animation Studios. *USD API Reference. UsdGeomMesh Class Reference*. URL: https://graphics.pixar.com/usd/release/api/class_usd_geom_mesh.html (zitiert auf S. 32, 33).

[Pix21a] Pixar Animation Studios. *Introduction to USD*. 2021. URL: https://graphics.pixar.com/usd/release/intro.html (zitiert auf S. 15, 27, 31).

[Pix21b] Pixar Animation Studios. *USD Frequently Asked Questions*. 2021. URL: https://graphics.pixar.com/usd/release/usdfaq.html (zitiert auf S. 27).

[Pix21c] Pixyz Software. *Pixyz Plugin for Unity 2020.2 Documentation - Meshes VS Nurbs*. 2021. URL: https://www.pixyz-software.com/documentations/html/2020.2/plugin4unity/MeshesVSNurbs.html (zitiert auf S. 17).

[Plu14] Pluralsight. *Key 3D Rigging Terms to Get You Moving*. 2014. URL: https://www.pluralsight.com/blog/film-games/key-rigging-terms-get-moving (zitiert auf S. 22).

[PR 11] PR Newswire Association LLC. *Lucasfilm and Sony Pictures Imageworks Release Alembic 1.0*. Aug. 2011. URL: https://web.archive.org/web/20171022193700/https://www.prnewswire.com/news-releases/lucasfilm-and-sony-pictures-imageworks-release-alembic-10-127316408.html (zitiert auf S. 26).

[Rea] Reallusion Inc. *What is IK/FK*. URL: https://www.reallusion.com/iclone/help/iclone4/pro/08_Animation/Motion_Layer/What_is_IK_FK.htm (zitiert auf S. 22).

[Red] M. Reddy. *B1. Object Files (.obj)*. URL: http://www.martinreddy.net/gfx/3d/OBJ.sp (zitiert auf S. 25).

[Rob21] Robert Cervellione. *Omniverse For AEC: Getting Started with McNeel Rhino*. Nov. 2021. URL: https://www.nvidia.com/en-us/on-demand/session/omniverse2020-om1403/ (zitiert auf S. 57, 58).

[Sid] SideFX. *Houdini 19.0 Documentation. USD Basics*. URL: hhttps://www.sidefx.com/docs/houdini/solaris/usd.html (zitiert auf S. 27).

[Son] Sony Pictures Imageworks Inc. and Industrial Light & Magic. *What is AbcMaterial?* URL: https://code.google.com/archive/p/alembic/wikis/AbcMaterial.wiki (zitiert auf S. 26).

[Son16] Sony Pictures Imageworks Inc. and Industrial Light & Magic. *Alembic 1.7.0*. 2012 - 2016. URL: http://docs.alembic.io (zitiert auf S. 26).

[The18] The Linux Foundation. *ACADEMY SOFTWARE FOUNDATION*. 2018. URL: https://www.aswf.io/ (zitiert auf S. 15).

[Uni] Universität Stuttgart Sonderforschungsbereich 1244. *Demonstrator*. URL: https://www.sfb1244.uni-stuttgart.de/demonstrator/ (zitiert auf S. 49).

[Uni17] Unity Technologies. *Unity Manual. Procedural Materials*. 2017. URL: https://docs.unity3d.com/550/Documentation/Manual/ProceduralMaterials.html (zitiert auf S. 21).

[Uni22a] Unity Technologies. *Unity Manual. Importing Assets*. 2022. URL: https://docs.unity3d.com/Manual/ImportingAssets.html (zitiert auf S. 15).

[Uni22b] Unity Technologies. *Unity Manual. Types of Light*. Jan. 2022. URL: https://docs.unity3d.com/Manual/Lighting.html (zitiert auf S. 20).

[Uni22c] Unity Technologies. *Unity Manual. Using Shader Graph*. 2022. URL: https://docs.unity3d.com/Manual/shader-graph.html (zitiert auf S. 20).

[Uni22d] Unity3d. *Eine kurze Einführung in den Unity Asset Store*. 2022. URL: https://unity3d.com/de/quick-guide-to-unity-asset-store (zitiert auf S. 11).

[Vil22] N. Villanueva. „What Is the 3D Production Pipeline and Why Is It Important?“ In: *Beginning 3D Game Assets Development Pipeline: Learn to Integrate from Maya to Unity*. Berkeley, CA: Apress, 2022, S. 1–30. ISBN: 978-1-4842-7196-4. DOI: 10.1007/978-1-4842-7196-4_1. URL: https://doi.org/10.1007/978-1-4842-7196-4_1 (zitiert auf S. 12, 15).

[Vis] Visualisierungsinstitut der Universität Stuttgart. *MegaMol. A cross-platform Visualization Framework*. URL: https://megamol.org/ (zitiert auf S. 31).

Alle URLs wurden zuletzt am 05. 03. 2022 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift